Trapping Rain Water in a Matrix

Difficulty Level : Hard • Last Updated : 01 Sep, 2021

Given a matrix **arr[][]** of dimension **M*N** consisting of positive integers, where **arr[i][j]** represents the height of each unit cell, the task is to find the total volume of water trapped in the matrix after rain.

Examples:

> ***Input:*** *arr[][] = {{4, 2, 7}, {2, 1, 10}, {5, 10, 2}}*
> ***Output:*** *1*
> ***Explanation:***
> *The rain water can be trapped in the following way:*
>
> 1. *The cells, {(0, 0), (0, 1), (0, 2), (1, 0), (1, 2), (2, 0), (2, 1), (2, 2)} traps 0 unit volume of rain water as all water goes out of the matrix as cells are on the boundary.*
> 2. *The cell (2, 2) traps 1 unit volume of rain water in between the cells {(0, 1), (1, 0), (1, 2), and (2, 1)}.*
>
> *Therefore, a total of 1 unit volume of rain water has been trapped inside the matrix.*

Recommended: Please try your approach on *{IDE}* first, before moving on to the solution.

Approach: The given problem can be solved by using the Greedy Technique and Min-Heap. Follow the steps below to solve the problem:

- Initialize a Min-Heap using the priority_queue, say **PQ**, to store the pairs of positions of a cell and its height.
- Push all the boundary cells in the **PQ** and mark all the pushed cells as visited.
- Initialize two variables, say **ans** as **0** and **maxHeight** as **0** to store the total volume and the maximum height of all the cells in **PQ** respectively.
- Iterate until **PQ** is not empty and perform the following steps:
  - Store the top node of **PQ** in a variable, say **front**, and erase the top element of **PQ**.
  - Update the value of **maxHeight** as the maximum of **maxHeight** and **front.height**.
  - Now, traverse to all the adjacent nodes of the current cell **(front.X, front.Y)** and do the following:
    - If the adjacent cell is valid i.e, the cell is not out of bound and not yet visited, then, push the value of the adjacent cell into **PQ.**
    - If the height of the adjacent cell is less than **maxHeight** then increment the **ans** by the difference of **maxHeight** and the height of the adjacent cell.
- Finally, after completing the above steps, print the value of **ans** as the resultant water trapped after rain.

Below is the implementation of the above approach:

```
// C++ program for the above approach
#include <bits/stdc++.h>
using namespace std;

// Stores the direction of all the
// adjacent cells
vector<vector<int> > dir
    = { { -1, 0 }, { 0, 1 }, { 1, 0 }, { 0, -1 } };

// Node structure
struct node {

    int height;
    int x, y;
};

// Comparator function to implement
// the min heap using priority queue
struct Compare {

    // Comparator function
    bool operator()(node const& a, node const& b)
    {
        return a.height > b.height;
    }
};

// Function to find the amount of water
// the matrix is capable to hold
int trapRainWater(vector<vector<int> >& heightMap)
{
    int M = heightMap.size();
    int N = heightMap[0].size();

    // Stores if a cell of the matrix
    // is visited or not
    vector<vector<bool> > visited(M,
                              vector<bool>(N, false));

    // Initialize a priority queue
    priority_queue<node, vector<node>, Compare> pq;
```

```
            // If element is not on
            // the boundary
            if (!(i == 0 || j == 0 || i == M - 1
                  || j == N - 1))
                continue;

            // Mark the current cell
            // as visited
            visited[i][j] = true;

            // Node for priority queue
            node t;
            t.x = i;
            t.y = j;
            t.height = heightMap[i][j];

            // Pushe all the adjacent
            // node in the pq
            pq.push(t);
        }
    }

    // Stores the total volume
    int ans = 0;

    // Stores the maximum height
    int max_height = INT_MIN;

    // Iterate while pq is not empty
    while (!pq.empty()) {

        // Store the top node of pq
        node front = pq.top();

        // Delete the top element of pq
        pq.pop();

        // Update the max_height
        max_height = max(max_height, front.height);
```

```
        for (int i = 0; i < 4; i++) {

            int new_x = curr_x + dir[i][0];
            int new_y = curr_y + dir[i][1];

            // If adjacent cells are out
            // of bound or already visited
            if (new_x < 0 || new_y < 0 || new_x >= M
                || new_y >= N || visited[new_x][new_y]) {
                continue;
            }

            // Stores the height of the
            // adjacent cell
            int height = heightMap[new_x][new_y];

            // If height of current cell
            // is smaller than max_height
            if (height < max_height) {

                // Increment the ans by
                // (max_height-height)
                ans = ans + (max_height - height);
            }

            // Define a new node
            node temp;
            temp.x = new_x;
            temp.y = new_y;
            temp.height = height;

            // Push the current node
            // in the pq
            pq.push(temp);

            // Mark the current cell
            // as visited
            visited[new_x][new_y] = true;
        }
    }
```

```
// Driver Code
int main()
{
    vector<vector<int> > arr = { { 1, 4, 3, 1, 3, 2 },
                                 { 3, 2, 1, 3, 2, 4 },
                                 { 2, 3, 3, 2, 3, 1 } };
    cout << trapRainWater(arr);

    return 0;
}
```

Output

```
4
```

Time Complexity: $(N * M * log(N * M))$

Auxiliary Space: $O(N * M)$

Like 3

Previous

Shortest Path Properties

Next

Convert A into B by incrementing or decrementing 1, 2, or 5 any number of times

RECOMMENDED ARTICLES

Page : 1 2 3

01 **Trapping Rain Water**
16, Aug 15

02 **Minimum number of Water to Land conversion to make two islands connected in a Grid**
30, Apr 20

03 **Maximum litres of water that can be bought with N Rupees**
21, Nov 18

04 **Implementing Water Supply Problem using Breadth First Search**
27, Apr 20

05 **Minimum sprinklers required to water a rectangular park**
03, Jun 20

06 **Count of operation required to water all the plants**
20, Jan 22

07 **Water Connection Problem**
12, Jan 18

08 **Maximum water that can be stored between two buildings**
31, May 19

feedback@geeksforgeeks.org

Article Contributed By :

surajanand
@surajanand

Vote for difficulty

Current difficulty : Hard

Company

About Us
Careers
In Media
Contact Us

Learn

Algorithms
Data Structures
SDE Cheat Sheet
Machine learning

Improved By : sagar0719kumar

Article Tags : min-heap, priority-queue, Arrays, Greedy, Heap, Matrix, Queue

Practice Tags : Arrays, Greedy, Matrix, Queue, Heap, priority-queue

Improve Article Report Issue

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Load Comments