542. 01 Matrix

April 21, 2017 | 70.6K views

Previous Next

Given a matrix consists of 0 and 1, find the distance of the nearest 0 for each cell.

The distance between two adjacent cells is 1.

Example 1:

```
Input:
[[0,0,0],
 [0,1,0],
 [0,0,0]]

Output:
[[0,0,0],
 [0,1,0],
 [0,0,0]]
```

Example 2:

```
Input:
[[0,0,0],
 [0,1,0],
 [1,1,1]]

Output:
[[0,0,0],
 [0,1,0],
 [1,2,1]]
```

Note:

1. The number of elements of the given matrix will not exceed 10,000.
2. There are at least one 0 in the given matrix.
3. The cells are adjacent in only four directions: up, down, left and right.

Solution

Approach #1 Brute force [Time Limit Exceeded]

Intuition

Do what the question says.

Algorithm

- Initialize `dist[i][j]=INT_MAX` for all `(i,j)` cells.
- Iterate over the matrix.
- If cell is `0`, `dist[i][j]=0`.
- Else, for each `1` cell,
  - Iterate over the entire matrix
  - If the cell is `0`, calculate its distance from current cell as `abs(k-i)+abs(l-j)`.
  - If the distance is smaller than the current distance, update it.

C++

```
vector<vector<int>> updateMatrix(vector<vector<int>>& matrix)
{
    int rows = matrix.size();
    if (rows == 0)
        return matrix;
    int cols = matrix[0].size();
    vector<vector<int>> dist(rows, vector<int>(cols, INT_MAX));
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            if (matrix[i][j] == 0)
                dist[i][j] = 0;
            else {
                for (int k = 0; k < rows; k++)
                    for (int l = 0; l < cols; l++)
                        if (matrix[k][l] == 0) {
                            int dist_kl = abs(k - i) + abs(l - j);
                            dist[i][j] = min(dist[i][j], dist_kl);
                        }
            }
        }
    }
    return dist;
}
```

Complexity Analysis

- Time complexity: $O((r \cdot c)^2)$. Iterating over the entire matrix for each `1` in the matrix.
- Space complexity: $O(r \cdot c)$. No extra space required than the `vector<vector<int> > dist`

Approach #2 Using BFS [Accepted]

Intuition

A better brute force: Looking over the entire matrix appears wasteful and hence, we can use Breadth First Search(BFS) to limit the search to the nearest `0` found for each `1`. As soon as a `0` appears during the BFS, we know that the `0` is nearest, and hence, we move to the next `1`.

Think again: But, in this approach, we will only be able to update the distance of one `1` using one BFS, which could in fact, result in slightly higher complexity than the Approach #1 brute force. But hey,this could be optimised if we start the BFS from `0`s and thereby, updating the distances of all the `1`s in the path.

Algorithm

- For our BFS routine, we keep a queue, `q`, to maintain the queue of cells to be examined next.
- We start by adding all the cells with `0`s to `q`.
- Initially, distance for each `0` cell is `0` and distance for each `1` is `INT_MAX`, which is updated during the BFS.
- Pop the cell from queue, and examine its neighbours. If the new calculated distance for neighbour `(i,j)` is smaller, we add `(i,j)` to `q` and update `dist[i][j]`.

C++

```
vector<vector<int>> updateMatrix(vector<vector<int>>& matrix)
{
    int rows = matrix.size();
    if (rows == 0)
      return matrix;
    int cols = matrix[0].size();
    vector<vector<int>> dist(rows, vector<int>(cols, INT_MAX));
    queue<pair<int, int>> q;
    for (int i = 0; i < rows; i++)
        for (int j = 0; j < cols; j++)
            if (matrix[i][j] == 0) {
                dist[i][j] = 0;
                q.push({ i, j }); //Put all 0s in the queue.
            }

    int dir[4][2] = { { -1, 0 }, { 1, 0 }, { 0, -1 }, { 0, 1 } };
    while (!q.empty()) {
        pair<int, int> curr = q.front();
        q.pop();
        for (int i = 0; i < 4; i++) {
            int new_r = curr.first + dir[i][0], new_c = curr.second + dir[i][1];
            if (new_r >= 0 && new_r < rows && new_c >= 0 && new_c < cols) {
                if (dist[new_r][new_c] > dist[curr.first][curr.second] + 1) {
                    dist[new_r][new_c] = dist[curr.first][curr.second] + 1;
                    q.push({ new_r, new_c });
                }
            }
        }
    }
```

Complexity analysis

- Time complexity: $O(r \cdot c)$.
- Since, the new cells are added to the queue only if their current distance is greater than the calculated distance, cells are not likely to be added multiple times.
- Space complexity: $O(r \cdot c)$. Additional $O(r \cdot c)$ for queue than in Approach #1

Approach #3 DP Approach [Accepted]

Intuition

The distance of a cell from `0` can be calculated if we know the nearest distance for all the neighbours, in which case the distance is minimum distance of any neighbour + 1. And, instantly, the word come to mind DP!!

For each `1`, the minimum path to `0` can be in any direction. So, we need to check all the 4 direction. In one iteration from top to bottom, we can check left and top directions, and we need another iteration from bottom to top to check for right and bottom direction.

Algorithm

- Iterate the matrix from top to bottom-left to right:
- Update $\text{dist}[i][j] = \min(\text{dist}[i][j], \min(\text{dist}[i][j-1], \text{dist}[i-1][j]) + 1)$ i.e., minimum of the current dist and distance from top or left neighbour + 1, that would have been already calculated previously in the current iteration.
- Now, we need to do the back iteration in the similar manner: from bottom to top-right to left:
- Update $\text{dist}[i][j] = \min(\text{dist}[i][j], \min(\text{dist}[i][j+1], \text{dist}[i+1][j]) + 1)$ i.e. minimum of current dist and distances calculated from bottom and right neighbours, that would be already available in current iteration.

C++

```
vector<vector<int>> updateMatrix(vector<vector<int>>& matrix)
{
    int rows = matrix.size();
    if (rows == 0)
        return matrix;
    int cols = matrix[0].size();
    vector<vector<int>> dist(rows, vector<int>(cols, INT_MAX - 100000));

    //First pass: check for left and top
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            if (matrix[i][j] == 0)
                dist[i][j] = 0;
            else {
                if (i > 0)
                    dist[i][j] = min(dist[i][j], dist[i - 1][j] + 1);
                if (j > 0)
                    dist[i][j] = min(dist[i][j], dist[i][j - 1] + 1);
            }
        }
    }

    //Second pass: check for bottom and right
    for (int i = rows - 1; i >= 0; i--) {
        for (int j = cols - 1; j >= 0; j--) {
            if (i < rows - 1)
                dist[i][j] = min(dist[i][j], dist[i + 1][j] + 1);
            if (j < cols - 1)
```

Complexity analysis

- Time complexity: $O(r \cdot c)$. 2 passes of $r \cdot c$ each
- Space complexity: $O(r \cdot c)$. No additional space required than `dist vector<vector<int> >`

Rate this article:

Previous Next

Comments: 34

Sort By

Type comment here... (Markdown is supported)

Preview Post

a-b-c ★ 692 April 15, 2019 1:21 AM

Why is this problem tagged as `Depth First Search` if DFS can't be used to solve it?

53 Share Reply

SHOW 3 REPLIES

lenchen1112 ★ 1007 October 21, 2018 8:07 PM

Python3 solution with only O(1) space complexity.
Scan twice from top-left and bottom-right separately.

```
class Solution:
    def updateMatrix(self, matrix: List[List[int]]) -> List[List[int]]:
```

Read More

13 Share Reply

SHOW 1 REPLY

ghostfacehillah ★ 52 April 3, 2018 11:35 AM

I think for approach #2, you actually could update the matrix directly.

9 Share Reply

SHOW 4 REPLIES

nate17 ★ 150 February 25, 2019 8:59 AM

Regarding to approach #3, I have a question, should not there be 4 pass updates in total, apart from the top left and bottom right direction, there are additional top right and bottom left directions?

5 Share Reply

SHOW 3 REPLIES

meatuiTk ★ 5 May 31, 2019 7:14 AM

Anyone who solved using DFS?

5 Share Reply

SHOW 6 REPLIES

wanders ★ 35 June 21, 2019 8:27 PM

Here's a level-set method in Python. It's basically BFS from multiple sources simultaneously, and in my opinion simpler than the other BFS approaches I've seen.

```
def updateMatrix(self, matrix):
```

Read More

3 Share Reply

SHOW 1 REPLY

lavoslifer ★ 19 May 30, 2020 4:52 AM

for approach 3: why do we do 2 passes? why not just 1 in all directions? is it in order to avoid an infinite loop?

2 Share Reply

SHOW 2 REPLIES

shiva700 ★ 23 April 29, 2020 12:32 PM

I did Approach 2 using DFS, but it is giving TLE, could someone help me with this?
Here is my code,

```
class Solution {
```

Read More

2 Share Reply

lidavet ★ 87 December 2, 2018 5:57 AM

I figured out the dynamic programming solution steps, but I didn't create a new matrix so it didn't work. Can someone explain why a new matrix is required for the 2 scan method?

2 Share Reply

SHOW 1 REPLY

hishank1996 ★ 1 June 1, 2020 9:12 PM

why two passes for the DP?

1 Share Reply

SHOW 2 REPLIES