问题描述

542.01矩阵

给定一个由0和1组成的矩阵,找出每个元素到最近的0的距离。

两个相邻元素间的距离为1。

示例 1:

输入:

| 0 0 0 | | | |
|-------|--|--|--|
| 0 1 0 | | | |
| 0 0 0 | | | |

输出:

```
0 0 0
0 1 0
0 0 0
```

示例 2:

输入:

```
0 0 0
0 1 0
1 1 1
```

输出:

```
0 0 0
0 1 0
1 2 1
```

注意:

- 1. 给定矩阵的元素个数不超过 10000。
- 2. 给定矩阵中至少有一个元素是 0。
- 3. 矩阵中的元素只在四个方向上相邻: 上、下、左、右。

解题思路

学习到了新知识:

#include<sstream>

可以引入新的类: istringstream. 这个类的作用是执行C++风格的串流的输入操作

例如:

```
string str="i am a boy";
  istringstream is(str);
  string s;
  while(is>>s)
  {
    cout<<s<<endl;
}</pre>
```

输出为:

i

am

a boy

会自动读取空格后。产生截断

回到本题中,我最开始选择使用DFS的方法解决:对于数值为0的元素距离赋0,对于1进行搜索。这样也是可以的的,但是我没能实现DFS。为了达到只遍历一遍就把遍历后的节点完成赋值,应该用BFS。

但是写到一半,我发现用动态规划更好;动态方程很容易写出,对于一个节点只需要从左上遍历一次,再从右下遍历一次即可:

动态方程写法是:对于从左上遍历,M[i][j]代表距离0的最近距离。

```
M(i)(j) = min(M[i][j], M[i-1][j], M[i][j-1])
```

同理对右下也如此即可。但是这样遍历的时间复杂度为: O(M*N) 即矩阵大小,空间复杂度为O(1);直接在原来矩阵上进行的存储。

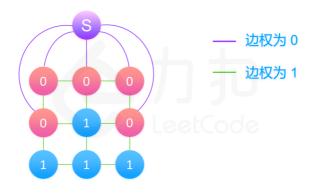
代码实现

```
#include<iostream>
#include<stdio.h>
#include<vector>
#include<queue>
#include<stack>
#include<string>
#include<sstream>
using namespace std;
class Solution {
public:
    vector<vector<int>> updateMatrix(vector<vector<int>>& matrix) {
        for (int i = 0; i < matrix.size(); i++)</pre>
            for (int j = 0; j < matrix[i].size(); j++)
                if (matrix[i][j] != 0)
                {
                    matrix[i][j] = INT_MAX;
                        matrix[i][j] = min(matrix[i-1][j] + 1, matrix[i][j]);
                    if (j > 0)
                        matrix[i][j] = min(matrix[i][j - 1] + 1, matrix[i][j]);
```

```
}
        }
        for(int i=matrix.size()-1;i>=0;i--)
             for (int j = matrix[i].size() - 1; j >= 0; j--)
             {
                 if (matrix[i][j] != 0)
                 {
                     if (i < matrix.size() - 1)</pre>
                         matrix[i][j] = min(matrix[i][j], matrix[i + 1][j] + 1);
                     if (j < matrix[i].size() - 1)</pre>
                         matrix[i][j] = min(matrix[i][j], matrix[i][j + 1] + 1);
                 }
            }
        return matrix;
    }
};
void input(vector<vector<int>> &Vec)
    int tmp = 0;
    string line;
    vector<int>v;
    while (!cin.eof())
        v.clear();
        getline(cin,line);
        istringstream ss(line);
        while (ss >> tmp)
            v.push_back(tmp);
        vec.push_back(v);
    }
}
int main()
{
    vector<vector<int>> M;
    input(M);
    Solution S;
    S.updateMatrix(M);
    for (int i=0;i<M.size();i++)</pre>
        for (int j = 0; j < M[i].size(); j++)
            cout << M[i][j] << ' ';</pre>
        cout << '\n';</pre>
    }
    return 0;
}
```

而使用BFS的方法:

先把distance=0 的加入节点,再逐层扩散,直到全部覆盖。相当于从一个虚构的 head节点出发,head下面一级全是0,0 下面一级全是1 类推扩散。图示:



```
# coding=utf-8
import collections
class Solution:
    def updateMatrix(self, matrix: List[List[int]]) -> List[List[int]]:
        m, n = len(matrix), len(matrix[0])
        Q = collections.deque([])
                                       # 取双边队列
        visited = set()
                                       # set()创建一个数据集, 其中的元素没有重复的
        # 初始化队列,加入m[i][j]=0的点
        for i in range(m):
            for j in range(n):
                if matrix[i][j] == 0:
                   Q.append((i,j))
                   visited.add((i,j))
        # BFS:
        while Q:
           i ,j = Q.popleft()
                                  # 取front
           for x, y in [(i+1, j), (i-1,j), (i, j+1), (i, j-1)]:
                if 0 \le x \le m and 0 \le y \le n and (x,y) not in visited:
                    matrix[x][y] = matrix[i][j] + 1
                   visited.add((x,y))
                   Q.append((x,y))
        return matrix
```