```
/*
在二维网格 grid 上，有 4 种类型的方格：

    1 表示起始方格。且只有一个起始方格。
    2 表示结束方格，且只有一个结束方格。
    0 表示我们可以走过的空方格。
    -1 表示我们无法跨越的障碍。

返回在四个方向（上、下、左、右）上行走时，从起始方格到结束方格的不同路径的数目，每一个无障碍方格都要通过一次。



示例 1：

输入：[[1,0,0,0],[0,0,0,0],[0,0,2,-1]]
输出：2
解释：我们有以下两条路径：
1. (0,0),(0,1),(0,2),(0,3),(1,3),(1,2),(1,1),(1,0),(2,0),(2,1),(2,2)
2. (0,0),(1,0),(2,0),(2,1),(1,1),(0,1),(0,2),(0,3),(1,3),(1,2),(2,2)

示例 2：

输入：[[1,0,0,0],[0,0,0,0],[0,0,0,2]]
输出：4
解释：我们有以下四条路径：
1. (0,0),(0,1),(0,2),(0,3),(1,3),(1,2),(1,1),(1,0),(2,0),(2,1),(2,2),(2,3)
2. (0,0),(0,1),(1,1),(1,0),(2,0),(2,1),(2,2),(1,2),(0,2),(0,3),(1,3),(2,3)
3. (0,0),(1,0),(2,0),(2,1),(2,2),(1,2),(1,1),(0,1),(0,2),(0,3),(1,3),(2,3)
4. (0,0),(1,0),(2,0),(2,1),(1,1),(0,1),(0,2),(0,3),(1,3),(1,2),(2,2),(2,3)

示例 3：

输入：[[0,1],[2,0]]
输出：0
解释：
没有一条路能完全穿过每一个空的方格一次。
请注意，起始和结束方格可以位于网格中的任意位置。



提示：

    1 <= grid.length * grid[0].length <= 20

来源：力扣（LeetCode）
链接：https://leetcode-cn.com/problems/unique-paths-iii
著作权归领扣网络所有。商业转载请联系官方授权，非商业转载请注明出处。
*/
```

分析

- 回溯法,利用经过0的此时做结束条件,同时建立visited标记数组来判断路径的合法性.

方法一:C++_回溯法

```cpp
class Solution
{

    private:
        void helper(    int& pathnum,
                        vector<vector<int>>&    grid            ,
                        int                     cur_row         ,
                        int                     cur_col         ,
                        int                     cur_num_zero    ,
                        int                     num_zeros       ,
                        vector<vector<int>>&    visited
        )
        {

            if(     cur_row < 0
                ||  cur_row >= grid.size()
                ||  cur_col < 0
                ||  cur_col >= grid[0].size()
            )
            {
                /* 范围越界 */
                return;
            }

            if(grid[cur_row][cur_col] == 2)
            {
                if (cur_num_zero == num_zeros)
                {
                    pathnum++;
                }
                else
                {
                    return;
                }
            }

            /*障碍物无法通过*/
            if(grid[cur_row][cur_col] == -1)
            {
                return;
            }

            /*回到起始位置了*/
            if(    grid[cur_row][cur_col]   ==  1
                && cur_num_zero             !=  0
            )
            {
                return;
            }


            // grid[cur_row][cur_col] == 0 或者 grid[cur_row][cur_col] == 1
```

```
            if (visited[cur_row][cur_col] == 1)
            {
                return;
            }
            else
            {
                visited[cur_row][cur_col] = 1;
                if(grid[cur_row][cur_col] == 0)    /**/
                {
                    cur_num_zero++;
                }
                /*上*/
                helper(pathnum, grid, cur_row - 1, cur_col, cur_num_zero , num_zeros, visited);
                /*下*/
                helper(pathnum, grid, cur_row + 1, cur_col, cur_num_zero , num_zeros, visited);
                /*左*/
                helper(pathnum, grid, cur_row, cur_col - 1, cur_num_zero , num_zeros, visited);
                /*右*/
                helper(pathnum, grid, cur_row, cur_col + 1, cur_num_zero , num_zeros, visited);
                visited[cur_row][cur_col] = 0;
            }
        }

    public:
        int uniquePathsIII(vector<vector<int>>& grid)
        {
            int start_row   =   -1              ;
            int start_col   =   -1              ;
            int num_zeros   =   0               ;

            int r           =   grid.size()     ;
            int c           =   grid[0].size()  ;

            int pathnum     =   0               ;

            vector<vector<int>>    visited = vector<vector<int>>(r, vector<int>(c, 0));

            for (int i = 0; i < r; i++)
            {
                for (int j = 0; j < c; j++)
                {
                    if (grid[i][j] == 0)
                    {
                        num_zeros++;
                    }
                    else if (grid[i][j] == 1)
                    {
                        start_row = i;
                        start_col = j;
                    }
                    // 貌似没用到
                    // int end_row = -1;
                    // int end_col = -1;
```

```
                    // else if (grid[i][j] == 2)
                    //{
                    //    end_row = i;
                    //    end_col = j;
                    //}
                }
            }
            helper(pathnum, grid, start_row, start_col, 0, num_zeros, visited);
            return pathnum;
        }
};



/*
执行结果:
通过
显示详情
执行用时 :4 ms, 在所有 cpp 提交中击败了89.61% 的用户
内存消耗 :8.4 MB, 在所有 cpp 提交中击败了94.12%的用户
*/
```

AlimyBreak
2019.12.02