```
/*
包含整数的二维矩阵 M 表示一个图片的灰度。你需要设计一个平滑器来让每一个单元的灰度成为平均灰度 (向下舍入) ，平均灰度的计算是周围的8个单元和它本身的值求平均，如果周围的单元格不足八个，则尽可能多的利用它们。
示例 1:
输入:
[[1,1,1],
 [1,0,1],
 [1,1,1]]
输出:
[[0, 0, 0],
 [0, 0, 0],
 [0, 0, 0]]
解释:
对于点 (0,0), (0,2), (2,0), (2,2): 平均(3/4) = 平均(0.75) = 0
对于点 (0,1), (1,0), (1,2), (2,1): 平均(5/6) = 平均(0.83333333) = 0
对于点 (1,1): 平均(8/9) = 平均(0.88888889) = 0
注意:
    给定矩阵中的整数范围为 [0, 255]。
    矩阵的长和宽的范围均为 [1, 150]。
来源：力扣（LeetCode）
链接：https://leetcode-cn.com/problems/image-smoother
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:

- 方法一:非原地解法:通过检测下标索引是否越界来计算总和和均值.
- 方法二:int的原地解法
  - 由于M的数据的范围是$0 \sim 255$,只用到int四个字节中的一个字节.
  - 我们可以从其他空闲字节中选取一个作为计算结果的存储空间,计算完毕后,把计算结果赋值给M即可.
  - 若传入的M是$vector < vector < unsignedchar >>$,该解法失效.
  - 与操作取低八位时默认了CPU是小端.

方法一:C++_非原地解法

```
class Solution
{

    private:
        int __neighborAver(vector<vector<int>>& M , int rows , int cols , int row , int col)
        {
            int temp    =   M[row][col];
            int count   =   1;

            /*上一行*/
            if(row-1>=0)
            {
                temp += M[row-1][col];
                count++;
```

```
                if(col-1 >=0)
                {
                    temp += M[row-1][col-1];
                    count++;
                }

                if(col+1 < cols)
                {
                    temp += M[row-1][col+1];
                    count++;
                }
            }

            /*下一行*/
            if(row+1< rows)
            {
                temp += M[row+1][col];
                count++;
                if(col-1 >=0)
                {
                    temp += M[row+1][col-1];
                    count++;
                }

                if(col+1 < cols)
                {
                    temp += M[row+1][col+1];
                    count++;
                }
            }

            /*中间一行*/
            if(col-1 >=0)
            {
                temp += M[row][col-1];
                count++;
            }

            if(col+1 < cols)
            {
                temp += M[row][col+1];
                count++;
            }

            return (int)(temp/count);
        }

    public:
        vector<vector<int>> imageSmoother(vector<vector<int>>& M)
        {
            int r   =   M.size();
            int c   =   M[0].size();

            vector<vector<int>> vvi(r,vector<int>(c,0));

            for(int i = 0; i < r; i++)
            {
                for(int j = 0 ; j < c; j++)
```

```
                {
                    vvi[i][j] = __neighborAver(M,r,c,i,j);
                }
            }
            return vvi;
        }
};



/*
执行结果:
通过
显示详情
执行用时 :168 ms, 在所有 cpp 提交中击败了97.11% 的用户
内存消耗 :17.4 MB, 在所有 cpp 提交中击败了88.18%的用户
*/
```

方法二:C++_原地解法

```
class Solution
{

    private:
        void __neighborAver(vector<vector<int>>& M , int rows , int cols , int row , int col)
        {
            int temp    =   (M[row][col] & 0xff);
            int count   =   1;

            /*上一行*/
            if(row-1>=0)
            {
                temp += (M[row-1][col] & 0xff);
                count++;
                if(col-1 >=0)
                {
                    temp += (M[row-1][col-1] & 0xff);
                    count++;
                }

                if(col+1 < cols)
                {
                    temp += (M[row-1][col+1] &  0xff);
                    count++;
                }
            }

            /*下一行*/
            if(row+1< rows)
            {
                temp += (M[row+1][col] & 0xff);
                count++;
                if(col-1 >=0)
                {
                    temp += (M[row+1][col-1] &  0xff);
```

```
                    count++;
                }

                if(col+1 < cols)
                {
                    temp += (M[row+1][col+1]& 0xff);
                    count++;
                }
            }

            /*中间一行*/
            if(col-1 >=0)
            {
                temp += (M[row][col-1]& 0xff);
                count++;
            }

            if(col+1 < cols)
            {
                temp += (M[row][col+1] & 0xff);
                count++;
            }

            (( unsigned char*)(&M[row][col]))[1] = temp/count;
            // //M[i][j] |= (temp/count)<<8;
        }

    public:
        vector<vector<int>> imageSmoother(vector<vector<int>>& M)
        {
            int r   =   M.size()    ;
            int c   =   M[0].size() ;
            int i   =   0           ;
            int j   =   0           ;

            /*计算8~15bit*/
            for(i = 0; i < r; i++)
            {
                for(j = 0 ; j < c; j++)
                {
                    __neighborAver(M,r,c,i,j);
                }
            }

            /*保留8~15bit--->0~7bit*/
            for(i = 0; i < r; i++)
            {
                for(j = 0 ; j < c; j++)
                {
                    //M[i][j] >>= 8;
                    M[i][j] = (( unsigned char*)(&M[i][j]))[1];
                }
            }


            return M;
        }
};
```

```
94
95  /*
96  执行结果:
97  通过
98  显示详情
99  执行用时 :264 ms, 在所有 cpp 提交中击败了31.23% 的用户
100 内存消耗 :17.6 MB, 在所有 cpp 提交中击败了79.09%的用户
101 https://leetcode-cn.com/submissions/detail/37351245/
102 */
```

AlimyBreak
2019.11.22