【华为OD机考 统一考试机试C卷】螺旋数字矩阵（C++ Java JavaScript Python C语言）

华为OD机考:统一考试 C卷 + D卷 + B卷 +A卷

目前在考C卷，经过两个月的收集整理，C卷真题已基本整理完毕
抽到原题的概率为2/3到3/3，也就是最少抽到两道原题。请注意：大家刷完C卷真题，最好要把B卷的真题刷一下，因为C卷的部分真题来自B卷。
另外订阅专栏还可以联系笔者开通在线 OJ 进行刷题，提高刷题效率。
真题目录：华为OD机考机试 真题目录（C卷 + D卷 + B卷 + A卷） + 考点说明
专栏：2023华为OD机试(B卷+C卷+D卷)（C++JavaJSPy）
华为OD面试真题精选：华为OD面试真题精选
在线OJ：点击立即刷题，模拟真实机考环境

题目描述

疫情期间，小明隔离在家，百无聊赖，在纸上写数字玩。他发明了一种写法：
给出数字个数n和行数m（0 < n ≤ 999，0 < m ≤ 999），从左上角的1开始，按照顺时针螺旋向内写方式，依次写出2,3…n，最终形成一个m行矩阵。
小明对这个矩阵有些要求：

- 每行数字的个数一样多
- 列的数量尽可能少
- 填充数字时优先填充外部
- 数字不够时，使用单个*号占位

输入描述

输入一行，两个整数，空格隔开，依次表示n、m

输出描述

符合要求的唯一矩阵

用例1

输入：

```
9 4
```

输出：

```
1 2 3
* * 4
9 * 5
8 7 6
```

说明：

> 9个数字写成4行，最少需要3列

用例2

输入：

3 5
输出：

```
1
2
3
*
*
```

说明：

> 3个数字写5行，只有一列，数字不够用*号填充

用例3

输入：

```
120 7
```

输出：

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 19
45 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 63 20
44 83 114 115 116 117 118 119 120 * * * * * * 99 64 21
43 82 113 112 111 110 109 108 107 106 105 104 103 102 101 100 65 22
42 81 80 79 78 77 76 75 74 73 72 71 70 69 68 67 66 23
41 40 39 38 37 36 35 34 33 32 31 30 29 28 27 26 25 24
```

解题思路

本题不难，主要就是模拟。按照题目的要求填充矩阵就可以了。

C++

```cpp
#include <iostream>
#include <vector>
#include <cmath>

int main() {
    int n, m;
    std::cin >> n >> m; // 读取要填充的数字个数n和矩阵的行数m

    int cols = std::ceil(static_cast<double>(n) / m); // 计算矩阵的列数
    std::vector<std::vector<int>> matrix(m, std::vector<int>(cols, 0)); // 创建一个整型矩阵，默认初始化为0

    int num = 1; // 用于填充的数字从1开始
    int top = 0, bottom = m - 1, left = 0, right = cols - 1;
    while (num <= n) {
        for (int i = left; i <= right && num <= n; i++) { // 从左到右填充上边界
            matrix[top][i] = num++;
        }

```

```
        top++; // 上边界下移
        for (int i = top; i <= bottom && num <= n; i++) { // 从上到下填充右边界
            matrix[i][right] = num++;
        }
        right--; // 右边界左移
        for (int i = right; i >= left && num <= n; i--) { // 从右到左填充下边界
            matrix[bottom][i] = num++;
        }
        bottom--; // 下边界上移
        for (int i = bottom; i >= top && num <= n; i--) { // 从下到上填充左边界
            matrix[i][left] = num++;
        }
        left++; // 左边界右移
    }

    for (int i = 0; i < m; i++) { // 遍历矩阵的每一行
        for (int j = 0; j < cols; j++) { // 遍历矩阵的每一列
            if (matrix[i][j] == 0) { // 如果当前位置是0，则输出'*'
                std::cout << '*';
            } else { // 否则输出当前位置的数字
                std::cout << matrix[i][j];
            }
            if (j < cols - 1) { // 在同一行的数字之间打印空格
                std::cout << " ";
            }
        }
        std::cout << std::endl; // 每打印完一行后换行
    }

    return 0;
}
```

Java

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int n = scanner.nextInt(); // 读取要填充的数字个数n
```

```java
        int m = scanner.nextInt(); // 读取矩阵的行数m
        scanner.close(); // 输入完毕后关闭scanner

        int cols = (int) Math.ceil(n / (double) m); // 计算矩阵的列数
        int[][] matrix = new int[m][cols]; // 创建一个整型矩阵，默认初始化为0

        int num = 1; // 用于填充的数字从1开始
        int top = 0, bottom = m - 1, left = 0, right = cols - 1;
        while (num <= n) {
            for (int i = left; i <= right && num <= n; i++) { // 从左到右填充上边界
                matrix[top][i] = num++;
            }
            top++; // 上边界下移
            for (int i = top; i <= bottom && num <= n; i++) { // 从上到下填充右边界
                matrix[i][right] = num++;
            }
            right--; // 右边界左移
            for (int i = right; i >= left && num <= n; i--) { // 从右到左填充下边界
                matrix[bottom][i] = num++;
            }
            bottom--; // 下边界上移
            for (int i = bottom; i >= top && num <= n; i--) { // 从下到上填充左边界
                matrix[i][left] = num++;
            }
            left++; // 左边界右移
        }

        for (int i = 0; i < m; i++) { // 遍历矩阵的每一行
            for (int j = 0; j < cols; j++) { // 遍历矩阵的每一列
                if (matrix[i][j] == 0) { // 如果当前位置是0，则输出'*'
                    System.out.print('*');
                } else { // 否则输出当前位置的数字
                    System.out.print(matrix[i][j]);
                }
                if (j < cols - 1) { // 在同一行的数字之间打印空格
                    System.out.print(" ");
                }
            }
            System.out.println(); // 每打印完一行后换行
        }
```

```
    }
}
```

javaScript

```javascript
const readline = require('readline').createInterface({
  input: process.stdin,
  output: process.stdout
});

readline.on('line', (input) => {
  const [n, m] = input.split(' ').map(Number); // 读取要填充的数字个数n和矩阵的行数m
  const cols = Math.ceil(n / m); // 计算矩阵的列数
  const matrix = Array.from({ length: m }, () => Array(cols).fill(0)); // 创建一个整型矩阵，默认初始化为0

  let num = 1; // 用于填充的数字从1开始
  let top = 0, bottom = m - 1, left = 0, right = cols - 1;
  while (num <= n) {
    for (let i = left; i <= right && num <= n; i++) { // 从左到右填充上边界
      matrix[top][i] = num++;
    }
    top++; // 上边界下移
    for (let i = top; i <= bottom && num <= n; i++) { // 从上到下填充右边界
      matrix[i][right] = num++;
    }
    right--; // 右边界左移
    for (let i = right; i >= left && num <= n; i--) { // 从右到左填充下边界
      matrix[bottom][i] = num++;
    }
    bottom--; // 下边界上移
    for (let i = bottom; i >= top && num <= n; i--) { // 从下到上填充左边界
      matrix[i][left] = num++;
    }
    left++; // 左边界右移
  }

  for (let i = 0; i < m; i++) { // 遍历矩阵的每一行
    let row = '';
    for (let j = 0; j < cols; j++) { // 遍历矩阵的每一列
      row += matrix[i][j] === 0 ? '*' : matrix[i][j]; // 如果当前位置是0，则输出'*'，否则输出当前位置的数字
```

```
      if (j < cols - 1) row += ' '; // 在同一行的数字之间添加空格
    }
    console.log(row); // 打印每一行
  }

  readline.close();
});
```

Python

```python
import math

n, m = map(int, input().split()) # 读取要填充的数字个数n和矩阵的行数m
cols = math.ceil(n / m) # 计算矩阵的列数
matrix = [[0 for _ in range(cols)] for _ in range(m)] # 创建一个整型矩阵，默认初始化为0

num = 1 # 用于填充的数字从1开始
top, bottom, left, right = 0, m - 1, 0, cols - 1
while num <= n:
    for i in range(left, right + 1): # 从左到右填充上边界
        if num <= n:
            matrix[top][i] = num
            num += 1
    top += 1 # 上边界下移
    for i in range(top, bottom + 1): # 从上到下填充右边界
        if num <= n:
            matrix[i][right] = num
            num += 1
    right -= 1 # 右边界左移
    for i in range(right, left - 1, -1): # 从右到左填充下边界
        if num <= n:
            matrix[bottom][i] = num
            num += 1
    bottom -= 1 # 下边界上移
    for i in range(bottom, top - 1, -1): # 从下到上填充左边界
        if num <= n:
            matrix[i][left] = num
            num += 1
    left += 1 # 左边界右移

```

```python
for row in matrix: # 遍历矩阵的每一行
    print(' '.join('*' if val == 0 else str(val) for val in row)) # 如果当前位置是0，则输出'*'，否则输出当前位置的数字
```

C语言

```c
#include <stdio.h>
#include <math.h>

int main() {
    int n, m;
    scanf("%d %d", &n, &m); // 读取要填充的数字个数n和矩阵的行数m

    int cols = (int)ceil((double)n / m); // 计算矩阵的列数
    int matrix[m][cols]; // 创建一个整型矩阵，默认初始化为0
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < cols; j++) {
            matrix[i][j] = 0;
        }
    }

    int num = 1; // 用于填充的数字从1开始
    int top = 0, bottom = m - 1, left = 0, right = cols - 1;
    while (num <= n) {
        for (int i = left; i <= right && num <= n; i++) { // 从左到右填充上边界
            matrix[top][i] = num++;
        }
        top++; // 上边界下移
        for (int i = top; i <= bottom && num <= n; i++) { // 从上到下填充右边界
            matrix[i][right] = num++;
        }
        right--; // 右边界左移
        for (int i = right; i >= left && num <= n; i--) { // 从右到左填充下边界
            matrix[bottom][i] = num++;
        }
        bottom--; // 下边界上移
        for (int i = bottom; i >= top && num <= n; i--) { // 从下到上填充左边界
            matrix[i][left] = num++;
        }
        left++; // 左边界右移
    }
```

```
    for (int i = 0; i < m; i++) { // 遍历矩阵的每一行
        for (int j = 0; j < cols; j++) { // 遍历矩阵的每一列
            if (matrix[i][j] == 0) { // 如果当前位置是0，则输出'*'
                printf("* ");
            } else { // 否则输出当前位置的数字
                printf("%d ", matrix[i][j]);
            }
        }
        printf("\n"); // 每打印完一行后换行
    }

    return 0;
}
```

文章目录


机考真题
华为OD
No No No
CSDN @算法大师