题目一（剑指offer04）

给定一个元素为非负整数的二维数组matrix，每行和每列都是从小到大有序的。再给定一个非负整数aim，请判断aim是否在matrix中

示例:

现有矩阵 matrix 如下：

```
[
  [1,   4,  7, 11, 15],
  [2,   5,  8, 12, 19],
  [3,   6,  9, 16, 22],
  [10, 13, 14, 17, 24],
  [18, 21, 23, 26, 30]
]
```

给定 target = 5，返回 true。

给定 target = 20，返回 false。

思路：

1. 从右上角开始寻找，如果cur小于target则row++，否则col--
2. 左边越界和下边越界则不存在；

```
var findNumberIn2DArray = function (matrix, target) {
    // 1. 从右上角开始寻找,如果cur小于target, j++, 如果cur大于target, i--
    if (matrix.length === 0 || matrix[0]?.length === 0) {
        return false;
    }
    let row = 0;
    let col = matrix[0].length - 1;
    while (row < matrix.length && col >= 0) {
        if (target === matrix[row][col]) {
            return true;
        } else if (target > matrix[row][col]) {
            row++;
        } else {
            col--;
        }
    }

    return false;
};
```

拓展： 每一行 0 肯定是会在 1 的左边，让你找到 matrix 里面含有最多 1 的那一行。

例如：

```
[
    [0,0,0,0,0,1,1,1,1],
    [0,0,0,0,0,0,0,1,1],
    [0,0,0,0,1,1,1,1,1],
    [0,0,0,0,1,1,1,1,1],
    [0,0,0,0,0,0,0,0,0],
    [0,1,1,1,1,1,1,1,1],
]
```

思路：

1. 从右上开始，向左如果next为1则继续，如果为0则向下；
2. 每一行都先向左尝试；如果越界则结束；

```
function findMaxRow(matrix) {
    if (matrix.length === 0 || matrix[0]?.length === 0) {
        return -1;
    }
    let maxRowOneCount = 0;
    let maxRow = -1;
    let col = matrix[0].length - 1;
    for (let row = 0; row < matrix.length; row++) {
        let ans = matrix[0].length - 1 - col;
        while (matrix[row][col] === 1) {
            ans++;
            if (col > 0 && matrix[row][col-1] === 0) {
                break;
            }
            col--;
        }
        maxRow = maxRowOneCount > ans ? maxRow : row;
        maxRowOneCount = maxRowOneCount > ans ? maxRowOneCount : ans;
    }
    return [maxRow, maxRowOneCount]
}
```

题目二

有n个打包机器从左到右一字排开，上方有一个自动装置会抓取一批放物品到每个打包机上，放到每个机器上的这些物品数量有多有少，由于物品数量不相同，需要工人将每个机器上的物品进行移动从而到达物品数量相等才能打包。

每个物品重量太大、每次只能搬一个物品进行移动，为了省力，只在相邻的机器上移动。物体从位置i搬动到位置j算一轮。

请计算在搬动最小轮数的前提下，使每个机器上的物品数量相等。

如果不能使每个机器上的物品相同，返回-1。

例如[1,0,5]表示有3个机器，每个机器上分别有1、0、5个物品，经过这些轮后：

第一轮：1 0 <- 5 => 1 1 4

第二轮：1 <-1<- 4 => 2 1 3

第三轮：2 1 <- 3 => 2 2 2

移动了3轮，每个机器上的物品相等，所以返回3

例如[2,2,3]表示有3个机器，每个机器上分别有2、2、3个物品，这些物品不管怎么移动，都不能使三个机器上物品数量相等，返回-1

思路（贪心）：

1. 计算每个机器上需要的物品数量（avg），如果不能等分返回1；
2. 遍历数组，假设当前下标为index，值为cur，计算index左边和右边整体分别多出的物品数leftMore和rightMore
    - 如果leftMore>0 rightMore>0，说明index左右两边物品都多了需要移动到index上，至少需要移动Max(leftMore, rightMore);
    - 如果leftMore<0 rightMore<0，说明index左右两边物品都缺少，需要从index移动到左右两边，至少需要abs(leftMore + rightMore);
    - 如果leftMore和rightMore一正一负，至少需要Max(abs(leftMore), abs(rightMore));

```
function packingMachine(arr) {
    if (arr === null || arr.length === 0) {
        return -1;
    }
    let sum = 0;
    let sumArr = [];
    let res = 0;
    for (let i = 0; i < arr.length; i++) {
        sum += arr[i];
        sumArr.push(sum)
    }
    if (sum % arr.length !== 0) {
        return -1;
    }
    let avg = sum / arr.length;

    for (let i = 0; i < arr.length; i++) {
        let leftMore = i === 0 ? 0 : sumArr[i-1] - avg * (i - 1);
        let rightMore = i === arr.length - 1 ? 0 : sum - sumArr[i] - avg *
(arr.length - i - 1);
        if (leftMore < 0 && rightMore < 0) {
            res = Math.max(res, Math.abs(leftMore+rightMore));
        } else {
            res = Math.max(res, Math.max(Math.abs(leftMore),
Math.abs(rightMore)));
        }
    }
```

```
    return res;
}
```

题目三（宏观调度）

用zigzag的方式打印矩阵，比如如下的矩阵

```
0 1 2 3
4 5 6 7
8 9 10 11
```

打印顺序为：0 1 4 8 5 2 3 6 9 10 7 11

思路：

1. 用两个点辅助，一个点向下向右，一个向右向下，两个点同步运动，每次打印两个点间的斜线上的数；

```
function zigzag(matrix) {
    let row = matrix.length;
    let col = matrix[0].length;

    let ar = 0;
    let ac = 0;
    let br = 0;
    let bc = 0;
    let deg = false;

    while (ac !== col) {
        printZigZag(ar, ac, br, bc, deg)
        ac = ar === row - 1 ? ac + 1 : ac;
        ar = ar === row - 1 ? ar : ar + 1;
        br = bc === col - 1 ? br + 1 : br;
        bc = bc === col - 1 ? bc : bc + 1;
        deg = !deg;
    }

    function printZigZag(ar, ac, br, bc, deg) {
        if (deg) {
            while (bc >= ac) {
                console.log(matrix[br++][bc--])
            }
        } else {
            while (ar >= br) {
                console.log(matrix[ar--][ac++])
            }
        }
    }
}
```

题目四

用螺旋的方式打印矩阵，比如下的矩阵

```
0 1 2 3
4 5 6 7
8 9 10 11
```

打印顺序为：0 1 2 3 7 11 10 9 8 4 5 6

思路：

1. 定位左上角a和右下角b可以形成一个正方形，顺时针打印这个正方形边框上的数；
2. 两个对角点向内移(ar++,ac++,br--,bc--)，顺时针打印边框；
3. 需要注意特殊状况ar等于br或者ac等于bc
4. 当ar>=br时停止；

```
function rotateMatrix(matrix) {
  let row = matrix.length;
  let col = matrix[0].length;

  let ar = 0;
  let ac = 0;
  let br = row - 1;
  let bc = col - 1;
  while (ar <= br && ac <= bc) {
    printEdge(ar++, ac++, br--, bc--);
  }

  function printEdge(ar, ac, br, bc) {
    if (ar === br) {
      for (let i = ac; i <= bc; i++) {
        console.log(matrix[ar][i]);
      }
    } else if (ac === bc) {
      for (let i = ar; i <= br; i++) {
        console.log(matrix[i][ac]);
      }
    } else {
      let curR = ar;
      let curC = ac;
      //  向右
      while (curC < bc) {
        console.log(matrix[curR][curC++]);
      }
      //  向下
      while (curR < br) {
        console.log(matrix[curR++][curC]);
      }
      // 向左
```

```
        while (curC > 0) {
          console.log(matrix[curR][curC--]);
        }
        // 向上
        while (curR > 0) {
          console.log(matrix[curR--][curC]);
        }
      }
    }
  }
```

题目五

给定一个正方形矩阵，只用有限几个变量，实现矩阵中每个位置的数顺时针转动90度，比如如下的矩阵

```
0  1  2  3
4  5  6  7
8  9  10 11
12 13 14 15
```

矩阵应该被调整为：

```
12 8  4 0
13 9  5 1
14 10 6 2
15 11 7 3
```

思路：

1. 用左上角a和右下角b确定一层正方形，一层层处理；
2. 将每一层正方形分成bc-ac组，如上矩阵可以分为3组

```
0        3


12       15
```

```
   1
         7
8
    14
```

```
      2
4
         11
   13
```

3. 遍历每一组，交换组内四个点的位置（matrix[ar][ac+i]、matrix[br-i][ac]、matrix[br][bc-i]、matrix[ar+i][bc]）

```
function printMatrixSpiralOrder(matrix) {
    const row = matrix.length;
    const col = matrix[0].length;

    let ar = 0;
    let ac = 0;
    let br = row - 1;
    let bc = col - 1;

    while (ar <= br) {
        spiralOrder(ar++, ac++, br--, bc--);
    }

    console.log(matrix)

    function spiralOrder(ar, ac, br, bc) {
        let step = br - ar;
        for (let i = 0; i < step; i++) {
            let temp = matrix[ar][ac+i]
            matrix[ar][ac+i] = matrix[br-i][ac]
            matrix[br-i][ac] = matrix[br][bc-i]
            matrix[br][bc-i] = matrix[ar+i][bc]
            matrix[ar+i][bc] = temp;
        }
    }
}
```

题目六

假设s和m初始化，s = "a"; m = s;

再定义两种操作，第一种操作：

```
m = s;
s = s + s;
```

第二种操作：

```
s = s + m;
```

求最小的操作步骤数，可以将s拼接到长度等于n

思路：

1. 如果n是质数，调用n-1次操作二是最小的；如果调用操作一会导致s中存在公因数（假设当前s=k，调用操作一后s=2k，m=k。之后无论怎么操作s中都包含公因数k，无法拼成n）
2. 如果n不是质数，则可以将n分解成质数相乘，假设可以分解成*xyzp，如果当前*=xy*z，在进行调用p-1次操作二可以完成目标；所以总操作数为x+y+z+p-4

```
function splitNbySm(n) {
  if (n < 2) {
    return 0;
  }
  if (isPrim(n)) {
    return n-1
  }
  const {sum, count} = divSumAndCount(n)
  return sum - count;
}

// 判断一个数是否是质数
function isPrim(n) {
  if (n < 2) {
    return false;
  }
  let max = Math.sqrt(n);
  for (let i = 2; i <= max; i++) {
    if (n % i === 0) {
      return false;
    }
  }
  return true;
}

/**
 *
 * @param {number} n 质数
 * @returns {object} sum: 因子和 count: 因子个数
 */
function divSumAndCount(n) {
  let sum = 0;
  let count = 0;
  for (let i = 2; i <= n; i++) {
    while ( n % i === 0) {
      sum += i;
      count++;
      n = n / i
    }
```

```
    }
    return {sum, count}
  }
```

题目七

给定一个字符串类型的数组arr，求其中出现次数最多的前K个

例子：

arr = ["a","b","c","a","x","y","a","b","c", "x", "x", "x"]

其中a3 b2 c2 x4 y1 出现次数最多的前2个为x4 a3

思路：

1. 用hash表记录词频表；
2. 用大根堆存储每个词频，以count排序

```
class Heap {
  constructor(cmp = (x, y) => x >= y) {
    this.cmp = cmp;
    this.heap = [];
  }
  insert(data) {
    const { cmp, heap } = this;
    heap.push(data);
    let index = this.size() - 1;
    let parentIndex = (index - 1) >> 1;
    while (index > 0 && cmp(heap[index], heap[parentIndex])) {
      this.swap(parentIndex, index);
      index = parentIndex;
      parentIndex = (index - 1) >> 1;
    }
  }
  pop() {
    const { heap } = this;
    if (this.isEmpty()) {
      return null;
    }
    this.swap(0, this.size() - 1);
    let res = heap.pop();
    this.heapfiy(0);

    return res;
  }
  heapfiy(index) {
    const { heap, cmp } = this;

    let left = 2 * index + 1;
```

```
    while (left < this.size()) {
      let maxIndex =
        left + 1 < this.size() && cmp(heap[left + 1], heap[left]) ? left + 1 :
left;
      if (cmp(heap[maxIndex], heap[index])) {
        this.swap(maxIndex, index);
        index = maxIndex;
        left = 2 * index + 1;
      }
    }
  }
  size() {
    return this.heap.length;
  }
  isEmpty() {
    return !this.heap.length;
  }
  peek() {
    return this.heap[0];
  }
  swap(i, j) {
    let temp = this.heap[i];
    this.heap[i] = this.heap[j];
    this.heap[j] = temp;
  }
}

function TimesNode(str, times) {
  this.str = str;
  this.times = times;
}

function TopKTimes(arr, k) {
  const hashMap = new Map();
  for (let i = 0; i < arr.length; i++) {
    if (hashMap.has(arr[i])) {
      hashMap.set(arr[i], hashMap.get(arr[i])  + 1);
    } else {
      hashMap.set(arr[i], 1);
    }
  }
  let heap = new Heap((x, y) => x.times >= y.times)
  for (let key of hashMap.keys()) {
    let node = new TimesNode(key, hashMap.get(key));
    heap.insert(node);
  }
  let res = [];
  for (let i = 0; i < k; i++) {
    res.push(heap.pop())
  }
  return res;
}
```

扩展：

设计结构，这个结构可以接受用户给的字符串add（str），且可以随时显示目前排名前k的字符串（动态结构）（大根堆/小根堆（门槛））

思路：改堆

词频表（key 字符串， value 词频）

小根堆[初始k个长度] heapsize = 0；以堆顶元素为门槛限制数据是否入堆；insert和heapfiy操作要更改位置map的记录

记录某个字符串在堆上的位置 map(key 字符串， value 位置)

1. 生成以上三个数据结构；
2. 当用户添加字符串时，更新词频表；
    - 如果位置 map显示在堆上，直接修改堆中数据，如果不在做如下判断；
    - 如果堆不满，将词频（node）加入堆（insert），位置map中记录词频；
    - 如果堆满，当前词频大于等于堆顶门槛，当前词频入堆，堆顶弹出，位置map中弹出词频位置标成-1；
    - 如果堆满，当前词频小于堆顶门槛，当前词频不入堆，位置map中位置标为-1；

```
class ModifyHeap {
  constructor(cmp = (x,y) => x <= y, positionMap) {
    this.cmp = cmp;
    this.heap = [];
    this.positionMap = positionMap;
  }
  modifyheap(i, times) {
    this.heap[i].times = times;
    this.heapfiy(i)
  }
  insert(data) {
    this.heap.push(data);
    this.positionMap.set(data.str, this.size() -1)
    let index = this.size() - 1;
    while (index) {
      let parentIndex = (index - 1) >> 1;
      if (!this.cmp(this.heap[parentIndex], this.heap[index])) {
        this.swap(index, parentIndex)
      }
      index = parentIndex;
    }
  }
  pop() {
    if (this.size() === 0) {
      return null;
    }
    this.swap(0, this.size() - 1);
    let res = this.heap.pop();
    this.positionMap.set(res, -1);
    this.heapfiy(0);
```

```
    return res;
  }
  heapfiy(index) {
    let left = 2 * index + 1;
    while (left < this.size()) {
      let lessIndex = left + 1 < this.size() && this.heap[left+1] <=
this.heap[left] ? left + 1 : left;
      if (!this.cmp(this.heap[index], this.heap[lessIndex])) {
        this.swap(lessIndex, index);
        index = lessIndex;
        left = 2 * index + 1;
      } else {
        break;
      }
    }
  }
  swap(i, j) {
    this.positionMap.set(this.heap[i].str, j)
    this.positionMap.set(this.heap[j].str, i)
    let temp = this.heap[i];
    this.heap[i] = this.heap[j];
    this.heap[j] = temp;
  }
  isEmpty() {
    return !this.heap.length;
  }
  size() {
    return this.heap.length;
  }
  peak() {
    return this.heap[0];
  }
}

class TopKDynamic {
  frequencyMap = new Map();
  positionMap = new Map();
  heap = new ModifyHeap((x, y) => x.times <= y.times, this.positionMap);

  constructor(k) {
    this.k = k;
  }

  add(str) {
    // 添加进词频表
    if (this.frequencyMap.has(str)) {
      this.frequencyMap.set(str, this.frequencyMap.get(str) + 1);
    } else {
      this.frequencyMap.set(str, 1);
    }
    if (this.positionMap.has(str) && this.positionMap.get(str) >= 0) {
      this.heap.modifyheap(this.positionMap.get(str), this.frequencyMap.get(str));
    } else {
      // 如果堆不满
```

```
        if (this.heap.size() < this.k) {
          this.heap.insert(new TimesNode(str, this.frequencyMap.get(str)));
        } else {
          // 如果当前词频大于门槛入堆
          if (this.frequencyMap.get(str) >= this.heap.peak().times) {
            this.positionMap.set(this.heap.pop().str, -1);
            this.heap.insert(new TimesNode(str,  this.frequencyMap.get(str)))
          } else {
            this.positionMap.set(str, -1);
          }
        }
      }

      console.log("Top3", this.heap.heap)
    }
  }
```