## spiralTraversal

### 문제

2차원 M x N 배열을 나선형(spiral)으로 순회해야 합니다.

### 입력

### 인자 1 : matrix

- 세로 길이(`matrix.length`)가 M, 가로 길이(`matrix[i].length`)가 N인 2차원 배열
- `matrix[i]`는 `string` 타입을 요소로 갖는 배열
- `matrix[i][j].length`는 1

### 출력

- `string` 타입을 리턴해야 합니다.

### 주의사항

- 순회는 좌측 상단 (0,0)에서 시작합니다.
- 배열의 모든 요소를 순서대로 이어붙인 문자열을 리턴해야 합니다.

### 입출력 예시

```
let matrix = [
  ['A', 'B', 'C'],
  ['D', 'E', 'F'],
  ['G', 'H', 'I'],
];
let output = spiralTraversal(matrix);
console.log(output); // --> 'ABCFIHGDE'

matrix = [
  ['T', 'y', 'r', 'i'],
  ['i', 's', 't', 'o'],
  ['n', 'r', 'e', 'n'],
  ['n', 'a', 'L', ' '],
];
output = spiralTraversal(matrix);
console.log(output); // --> 'Tyrion Lannister'
```

```javascript
const spiralTraversal = function (matrix) {
  // 각 방향마다 row와 col의 변화를 저장
  const RIGHT = [0, 1];
  const DOWN = [1, 0];
  const LEFT = [0, -1];
  const UP = [-1, 0];
  // 각 방향을 위한 lookup table
  const MOVES = [RIGHT, DOWN, LEFT, UP];
  const M = matrix.length;
  const N = matrix[0].length;
  const isValid = (row, col) => row >= 0 && row < M && col >= 0 && col < N;

  let cnt = 0;
  let row = 0,
    col = -1;
  let direction = 0;
  let result = '';
  while (cnt < M * N) {
    const move = MOVES[direction];
    const [rd, cd] = move;

    row = row + rd;
    col = col + cd;
    while (isValid(row, col) && matrix[row][col] !== false) {
      result = result + matrix[row][col];
      // 한 요소를 두 번 접근하지 않게 하기 위해서, 접근된 요소를 false로 변경한다.
      matrix[row][col] = false;
      row = row + rd;
      col = col + cd;
      cnt++;
    }
    // row, col 이 행렬의 범위를 벗어났기 때문에,
    // 진행된 방향의 반대로 한 칸 이동한다.
    row = row - rd;
    col = col - cd;
    // 각 방향이 순환되기 때문에 모듈러 연산을 사용한다.
    direction = (direction + 1) % 4;
  }
  return result;
};
```

## orderOfPresentation

### 문제

말썽꾸러기 김코딩은 오늘도 장난을 치다가 조별 발표 순서가 담긴 통을 쏟고 말았습니다.

선생님께서는 미리 모든 발표 순서의 경우의 수를 저장해 놓았지만 김코딩의 버릇을 고치기 위해 문제를 내겠다고 말씀하셨습니다.

김코딩은 모든 조별 발표 순서에 대한 경우의 수를 차례대로 구한 뒤 발표 순서를 말하면 이 발표 순서가 몇 번째 경우의 수인지를 대답해야 합니다.

총 조의 수 N과 선생님이 말씀하시는 발표 순서 k가 주어질 때, 김코딩이 정답을 말 할 수 있게 올바른 리턴 값을 구하세요.

모든 경우의 수가 담긴 배열은 번호가 작을수록 앞에 위치한다고 가정합니다.

ex) N = 3일경우, [[1,2,3],[1,3,2],[2,1,3],[2,3,1],[3,1,2],[3,2,1]]

### 입력

### 인자 1: N

- `Number` 타입의 1 <= N <= 12인 조의 개수

### 인자 2: K

- `Number`타입의 Array (0 <= index)

ex) n이 3이고 k가 [2, 3, 1]일 경우

모든 경우의 수를 2차원 배열에 담는다면 [[1,2,3],[1,3,2],[2,1,3],[2,3,1],[3,1,2],[3,2,1]]이 되고,

반환하는 값은 3이 됩니다.

### 주의사항

- k내에 중복되는 요소는 없다고 가정합니다.

### 입출력 예시

```
let output = orderOfPresentation(3, [2, 3, 1]);
console.log(output); // 3

output = orderOfPresentation(5, [1, 3, 2, 4, 5])
console.log(output); // 6
```

```javascript
function orderOfPresentation(N, K) {
  // 조의 개수 N, 발표 순서 K

  // N은 최대 12입니다.
  // 발표 순서를 만드는 것은 순열(permutation)이므로, 발표 순서의 최대 크기는 12!입니다.
  // 이는 약 4억 8천만에 해당하며, 일반적인 수행 속도 상한(약 1억)을 훨씬 상회하므로 순열을 전부 생성하는 것은 올바른 접근 방법이 아닙니다.

  const factorial = (n) => {
    if (n <= 1) return 1;
    return n * factorial(n - 1);
  };

  // 발표 순서를 담는 변수 생성
  let order = 0;

  // N개의 조 중에, 어떠한 조가 이미 포함되었는지 확인하기 위해 배열을 생성합니다.
  // 만약 N이 3이라면 [false, false, false, false]로 생성됩니다.
  // 제일 첫 번째는 더미 데이터입니다. (인덱스는 0부터 시작하지만 조는 1부터 시작하기 때문에)
  const isUsed = Array(N + 1).fill(false);

  // K의 길이만큼 순회합니다.
  for (let i = 0; i < K.length; i++) {
    // K의 i번째 조를 변수에 담습니다.
    const num = K[i];
    // 사용했는지 판별하기 위해 isUsed에 체크합니다. (중복이 아니기 때문에)
    isUsed[num] = true;
    // num보다 앞에 올 수 있는 수들의 배열을 복제해서,
    const candidates = isUsed.slice(1, num);
    // 이 중에서 아직 사용되지 않은 수의 개수를 구합니다.
    const validCnt = candidates.filter((el) => el === false).length;
    // 아직 사용되지 않은 수, 그 전까지의 모든 경우의 수를 카운트합니다.
    const formerCnt = validCnt * factorial(N - i - 1);
    // order에 추가합니다.
    order = order + formerCnt;

    /**
     * 설명을 덧붙이자면,
     * 만약 K가 [2, 3, 1]이라고 가정했을 때, 첫 번째 num은 2가 될 것입니다.
     * 2가 제일 앞에 있다고 가정한다면, 앞자리가 2의 차례가 오기 전에 1의 모든 경우의 수를 구했을 것이고,
     * 1의 모든 경우의 수를 지금부터 구하게 됩니다.
     *
     * 그렇다면, IsUsed 배열은 이렇게 됩니다. [false(더미), false, true, false]
     * candidates 배열은 이렇게 됩니다. => [false]
     * validCnt는 이렇게 됩니다. => 1
     * formerCnt는 이렇게 됩니다. => 1 * factorial(3 - 0 - 1) // i는 0부터 시작하기 때문에 N에서 남아 있는 수를 구할 때 - 1이 추가로 필요합니다.
     * order는 2를 추가합니다.
     *
     * 두 번째를 순회했을 땐, num이 3이 됩니다.
     * 그렇다면, IsUsed 배열은 이렇게 됩니다. [false(더미), false, true, true]
     * candidates 배열은 이렇게 됩니다. => [false]
     * validCnt는 이렇게 됩니다 => 1
     * formerCnt는 이렇게 됩니다 => 1 * factorial(3 - 1 - 1)
     * order는 1을 추가합니다. (3)
     *
     * 세 번째를 순회했을 땐, num이 1이 됩니다.
     * IsUsed 배열은 이렇게 됩니다. [false, true, true, true]
     * candidates 배열은 []이고, validCnt는 0이 되어, formerCnt는 0이 됩니다.
     *
     * 발표 순서는 0부터 시작하기 때문에 0, 1, 2, 3으로
     * 결과적으로, 값은 3이 됩니다.
     */
  }

  return order;
}
```