329. Caminho crescente mais longo em uma matriz

Descrição do problema

Dada uma matriz de inteiros m x n, retorne o comprimento do caminho crescente mais longo na matriz.

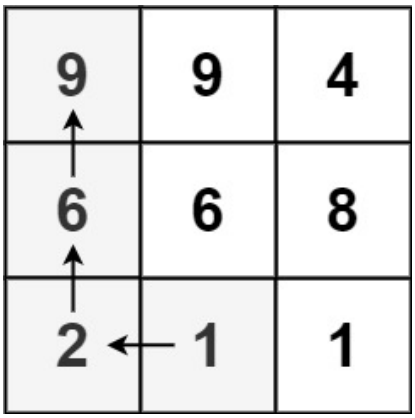
A partir de cada célula, você pode mover-se em quatro direções: esquerda, direita, para cima ou para baixo. Você não pode se mover na diagonal ou sair dos limites (ou seja, não é permitido contornar).

Restrições

- m == matrix.length
- n == matrix[i].length
- 1 <= m, n <= 200
- 0 <= matrix[i][j] <= 231 - 1

Exemplos:

Exemplo 1:



Entrada:

matrix = [[9,9,4],[6,6,8],[2,1,1]]

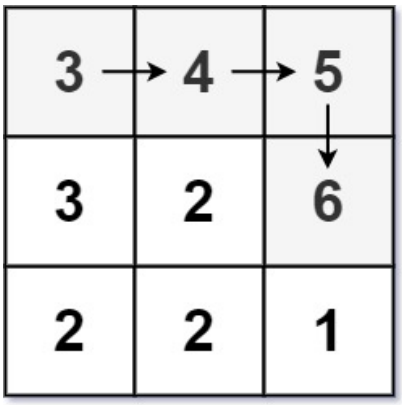
Saída:

4

Explicação:

> O Caminho crescente mais longo é [1, 2, 6, 9]

Exemplo 2:



Entrada:

matrix = [[3,4,5],[3,2,6],[2,2,1]]

Saída:

4

Explicação:

> O caminho crescente mais longo é [3, 4, 5, 6]. Mover nas diagonais não é permitido.

Solução proposta:

Para soluções em C++, o LeetCode oferece o seguinte ponto de partida:

```
class Solution:
    def longestIncreasingPath(self, matrix: List[List[int]]) -> int:
```

Ou seja, ele requer uma classe "solução", a qual contenha uma função que retorne um número inteiro, indicando o tamanho do maior caminho crescente dentro de uma matriz.

Começando a resolução do problema, torna-se intrinsecamente importante identificar que, caso a matriz não exista e/ou seja nula, devemos retornar 0. Este passo também se torna importante se a primeira linha é vazia.

Definido isso, usamos r e c para identificar as dimensões da matriz, e o vetor distance a fim de armazenar o maior caminho crescente possível que começa em cada célula, com os valores nesse vetor sendo inicializados em 0.

A fim de buscar o caminho crescente mais longo, a DFS foi o algoritmo de busca em grafos escolhido para a resolução deste problema.

O primeiro passo dentro da DFS é verificar se o valor da célula no vetor distance já foi calculado a fim de evitar que processamentos desnecessários ocorram.

Logo em seguida, a lógica para as direções é incluída no código. Como podemos "andar" apenas para a esquerda, direita, para cima e para baixo, uma variável contendo a lógica necessária para cada passo é incluída dentro do código da busca em profundidade.

Torna-se importante ressaltar que a variável max_path é inicializada com o valor 1 devido ao menor caminho crescente mais longo em uma matriz podendo ser exatamente o número 1.

Agora explorando as direções, caso alguma seja válida, a função DFS é chamada para calcular o caminho a partir da célula atual, comparando com o valor atual e garantindo que sempre estamos considerando o maior caminho possível.

Depois de explorar todas as direções, o valor max_path é guardado no vetor distance e retornado.

Assim, para que a "mágica" aconteça, a variável path é inicializada e percorremos a matriz com dois loops for e, para cada célula nas posições i e j, a função DFS é chamada, comparando com o valor atual de path.

Por fim, path é retornada.