



INSTITUTO POLITÉCNICO NACIONAL
Escuela Superior de Cómputo



Análisis y diseño de algoritmos

Ejercicio 11

Gil Juárez Hector David

3CV1

04 de junio del 2024

Problema

Dado un laberinto representado por una matriz $m \times n$ donde:

- 0 indica una celda libre.
- 1 indica un obstáculo. Debemos encontrar el número de caminos posibles desde la posición inicial (0, 0) hasta la posición final (m-1, n-1), moviéndonos solo hacia la derecha o hacia abajo.

Enfoque de Programación Dinámica

La programación dinámica es una técnica para resolver problemas complejos dividiéndolos en subproblemas más pequeños y solucionándolos una sola vez, almacenando sus soluciones.

Pasos del Enfoque

Definición del Estado

$dp[i][j]$ representa el número de caminos posibles para llegar a la celda (i, j) desde la celda (0, 0).

Inicialización

- Si la celda de inicio (0, 0) es un obstáculo ($laberinto[0][0] == 1$), no hay caminos posibles: retornamos 0.
- De lo contrario, inicializamos $dp[0][0] = 1$ porque hay una única manera de estar en la celda de inicio.

Llenado de la Primera Fila y Primera Columna

Para la primera fila y la primera columna, si una celda no es un obstáculo, puede ser alcanzada solo desde la celda anterior en la misma fila o columna.

Transición

Para cada celda (i, j) no bloqueada ($laberinto[i][j] == 0$), el número de caminos para llegar allí es la suma de los caminos para llegar a la celda de arriba (i-1, j) y la celda de la izquierda (i, j-1), siempre y cuando estas celdas no sean obstáculos.

Resultado

El valor en $dp[m-1][n-1]$ nos dará el número de caminos posibles para llegar a la celda final (m-1, n-1).

Código

```
def count_paths(laberinto):
    m = len(laberinto)
    n = len(laberinto[0])
    dp = [[0] * n for _ in range(m)]

    if laberinto[0][0] == 1 or laberinto[m-1][n-1] == 1:
        return 0

    dp[0][0] = 1

    for i in range(1, m):
        if laberinto[i][0] == 0:
            dp[i][0] = dp[i-1][0]

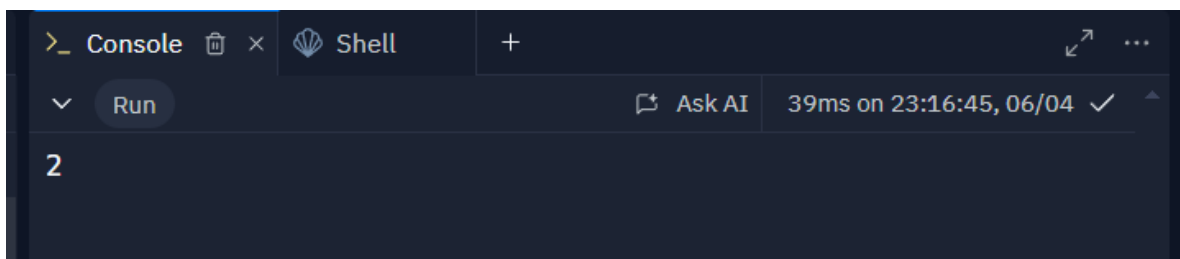
    for j in range(1, n):
        if laberinto[0][j] == 0:
            dp[0][j] = dp[0][j-1]

    for i in range(1, m):
        for j in range(1, n):
            if laberinto[i][j] == 0:
                if laberinto[i-1][j] == 0:
                    dp[i][j] += dp[i-1][j]
                if laberinto[i][j-1] == 0:
                    dp[i][j] += dp[i][j-1]

    return dp[m-1][n-1]

laberinto = [
    [0, 0, 0],
    [0, 1, 0],
    [0, 0, 0] ]
print(count_paths(laberinto))
```

Ejecución



The screenshot shows a code execution interface with a dark theme. At the top, there are tabs for 'Console' (active), 'Shell', and a '+' icon. Below the tabs, there is a 'Run' button and a status bar indicating '39ms on 23:16:45, 06/04' with a checkmark. The output area displays the number '2'.

Complejidad

La complejidad temporal y espacial de este algoritmo es $O(m \cdot n)$, donde m es el número de filas y n es el número de columnas de la matriz. Esto se debe a que recorreremos cada celda una sola vez y almacenamos los resultados intermedios en la matriz dp .