Profundiza más

Recurso de Profundización

Clase 14: Arreglos y funciones (Parte 2)

Autor: Damián Nicolalde Rodríguez

Objetivo del Documento:

El objetivo de este recurso de profundización es proporcionar una comprensión detallada de dos conceptos clave en programación utilizando recursión: búsqueda binaria recursiva y resolución de laberintos con recursión. A través de ejemplos prácticos y explicaciones paso a paso, los estudiantes aprenderán cómo implementar la recursión para resolver problemas como la búsqueda eficiente de elementos en una lista ordenada y la navegación de un laberinto. Además, se destacará la importancia de la recursión para la descomposición de problemas complejos en subproblemas más simples y manejables, así como las mejores prácticas y posibles optimizaciones en el uso de funciones recursivas.
Este recurso tiene como objetivo que los estudiantes:

1. Comprendan la mecánica de la recursión y su aplicabilidad en problemas clásicos de programación.
2. Desarrollen habilidades para implementar la búsqueda binaria recursiva como un ejemplo de algoritmo eficiente.
3. Aprendan a resolver problemas de backtracking, como la navegación de laberintos, utilizando recursión.
4. Optimicen la recursión mediante estrategias como la memorización para mejorar el rendimiento en algoritmos recursivos.

1. Búsqueda Binaria Recursiva

La **búsqueda binaria** es un algoritmo eficiente para encontrar un elemento dentro de una lista ordenada. En cada paso, el algoritmo divide la lista en dos mitades y compara el valor que estamos buscando con el valor en el medio de la lista. Según el resultado de esa comparación, el algoritmo sigue buscando en la mitad izquierda o en la mitad derecha de la lista, y repite el proceso recursivamente.

Profundiza más

Ventajas

- **Eficiencia**: La búsqueda binaria tiene una complejidad de tiempo de O(log n), lo que la hace mucho más eficiente que la búsqueda lineal, que tiene una complejidad de O(n).
- **Aplicación**: Es útil cuando tenemos grandes cantidades de datos ordenados y necesitamos realizar búsquedas rápidas.

Código Comentado: Búsqueda Binaria Recursiva

```
def busqueda_binaria(arr, objetivo, inicio, fin):
    # Caso base: Si el rango de búsqueda es válido
    if inicio > fin:
        return -1  # No encontrado

    # Encuentra el índice medio
    medio = (inicio + fin) // 2

    # Si el elemento está en el medio, retornamos su índice
    if arr[medio] == objetivo:
        return medio
    # Si el objetivo es menor que el elemento del medio, buscamos en la
mitad izquierda
    elif arr[medio] > objetivo:
        return busqueda_binaria(arr, objetivo, inicio, medio - 1)
    # Si el objetivo es mayor, buscamos en la mitad derecha
    else:
        return busqueda_binaria(arr, objetivo, medio + 1, fin)

# Lista ordenada
arr = [1, 3, 5, 7, 9, 11, 13, 15, 17, 19]
objetivo = 7

# Llamada a la función recursiva con el índice inicial 0 y el índice
final de la lista
resultado = busqueda_binaria(arr, objetivo, 0, len(arr) - 1)

# Imprimir el resultado
if resultado != -1:
    print(f"Elemento {objetivo} encontrado en el índice {resultado}.")
else:
    print(f"Elemento {objetivo} no encontrado en la lista.")
```

Profundiza más

Explicación del Código

- **Caso Base**: Si el inicio es mayor que el fin, eso significa que hemos agotado todas las opciones y el número no está presente en la lista, por lo que retornamos -1.
- **Recursión**: Si el número no es el elemento del medio, entonces compararemos si el número objetivo es menor o mayor que el valor medio:
    - Si es menor, realizamos la búsqueda en la mitad izquierda.
    - Si es mayor, realizamos la búsqueda en la mitad derecha.

Salida Esperada

```
Elemento 7 encontrado en el índice 3.
```

2. Resolución de Laberintos con Recursión

Los laberintos son estructuras que se resuelven de manera natural utilizando recursión. En un laberinto, hay caminos que nos llevan a una salida, y la recursión nos ayuda a explorar esos caminos de forma eficiente. Usamos una estrategia llamada **backtracking**, en la que intentamos avanzar por un camino, pero si nos encontramos con un callejón sin salida, retrocedemos y probamos otro camino.

Código Comentado: Resolución de Laberinto con Recursión

En este ejemplo, resolveremos un laberinto de 0s (camino libre) y 1s (pared) en una cuadrícula, donde tenemos que encontrar la salida.

```
def resolver_laberinto(maze, x, y, solucion):
    # Caso base: Si (x, y) es fuera de los límites del laberinto o es una
pared
    if x < 0 or y < 0 or x >= len(maze) or y >= len(maze[0]) or
maze[x][y] == 1:
        return False

    # Si llegamos al final (esquina inferior derecha), hemos resuelto el
laberinto
    if x == len(maze) - 1 and y == len(maze[0]) - 1:
        solucion[x][y] = 1  # Marca el camino hacia la salida
        return True

    # Marca el camino actual como parte de la solución
    solucion[x][y] = 1
```

Profundiza más

```
    # Recursivamente intenta movernos hacia abajo, derecha, arriba y
izquierda
    if resolver_laberinto(maze, x + 1, y, solucion):  # Moverse hacia
abajo
        return True
    if resolver_laberinto(maze, x, y + 1, solucion):  # Moverse hacia la
derecha
        return True
    if resolver_laberinto(maze, x - 1, y, solucion):  # Moverse hacia
arriba
        return True
    if resolver_laberinto(maze, x, y - 1, solucion):  # Moverse hacia la
izquierda
        return True

    # Si ninguno de los movimientos es válido, retrocedemos
    solucion[x][y] = 0
    return False

# Definir un laberinto
laberinto = [
    [0, 0, 1, 0, 0],
    [1, 0, 1, 0, 1],
    [1, 0, 0, 0, 0],
    [1, 1, 1, 1, 0],
    [0, 0, 0, 1, 0]
]

# Crear una solución vacía
solucion = [[0 for _ in range(len(laberinto[0]))] for _ in
range(len(laberinto))]

# Llamada inicial a la función recursiva
if resolver_laberinto(laberinto, 0, 0, solucion):
    print("Solución encontrada:")
    for fila in solucion:
        print(fila)
else:
    print("No se encontró solución.")
```

Explicación del Código

- **Caso Base**: Si la posición (x, y) es inválida (fuera de los límites del laberinto o una pared), retornamos False. Si llegamos a la posición final (len(maze) - 1, len(maze[0]) - 1), hemos encontrado la salida.
- **Recursión**: Intentamos movernos en las cuatro direcciones posibles (abajo, derecha, arriba, izquierda). Si alguna de las direcciones lleva a la solución, retornamos True.
- **Backtracking**: Si no encontramos una solución en el camino actual, marcamos la celda como 0 (vacía) para intentar otro camino.

Salida Esperada

```
Solución encontrada:
[1, 1, 0, 0, 0]
[0, 1, 0, 0, 0]
[0, 1, 1, 0, 0]
[0, 0, 0, 1, 0]
[0, 0, 0, 0, 1]
```

Este resultado muestra el camino desde la entrada hasta la salida del laberinto, donde los 1s indican el camino tomado.