Guía Completa de Algoritmos

Esta guía contiene explicaciones detalladas y ejemplos prácticos de todos los algoritmos esenciales para backend y desarrollo general.

1. Búsqueda Binaria

Descripción: La búsqueda binaria encuentra un elemento en una lista ordenada dividiendo el rango de búsqueda a la mitad en cada paso.

Ejemplo en Python

```
def busqueda_binaria(arr, x):
    inicio, fin = 0, len(arr) - 1
    while inicio <= fin:
        medio = (inicio + fin) // 2
        if arr[medio] == x:
            return medio
        elif arr[medio] < x:
            inicio = medio + 1
        else:
            fin = medio - 1
    return -1

# Ejemplo
lista = [1, 3, 5, 7, 9]
print(busqueda_binaria(lista, 7))  # Salida: 3
```

2. QuickSort

Descripción: Divide la lista en sublistas usando un pivote y las ordena recursivamente.

Ejemplo en Python

```
def quicksort(arr):
    if len(arr) <= 1:
        return arr
```

```
    pivote = arr[len(arr) // 2]
    izquierda = [x for x in arr if x < pivote]
    centro = [x for x in arr if x == pivote]
    derecha = [x for x in arr if x > pivote]
    return quicksort(izquierda) + centro + quicksort(derecha)

# Ejemplo
lista = [3, 6, 8, 10, 1, 2, 1]
print(quicksort(lista))  # Salida: [1, 1, 2, 3, 6, 8, 10]
```

3. BFS (Breadth-First Search)

Descripción: Explora nodos nivel por nivel en un grafo o árbol.

Ejemplo en Python

```
from collections import deque

def bfs(grafo, inicio):
    visitados = set()
    cola = deque([inicio])
    while cola:
        nodo = cola.popleft()
        if nodo not in visitados:
            visitados.add(nodo)
            cola.extend(grafo[nodo] - visitados)
    return visitados

# Ejemplo
grafo = {
    'A': {'B', 'C'},
    'B': {'A', 'D', 'E'},
    'C': {'A', 'F'},
    'D': {'B'},
    'E': {'B', 'F'},
    'F': {'C', 'E'}
}
print(bfs(grafo, 'A'))  # Salida: {'A', 'B', 'C', 'D', 'E', 'F'}
```

4. Dijkstra

Descripción: Encuentra la ruta más corta desde un nodo inicial a todos los demás en un grafo ponderado.

Ejemplo en Python

```
import heapq

def dijkstra(grafo, inicio):
    distancias = {nodo: float('infinity') for nodo in grafo}
    distancias[inicio] = 0
    prioridad = [(0, inicio)]

    while prioridad:
        distancia_actual, nodo_actual = heapq.heappop(prioridad)
        for vecino, peso in grafo[nodo_actual].items():
            distancia = distancia_actual + peso
            if distancia < distancias[vecino]:
                distancias[vecino] = distancia
                heapq.heappush(prioridad, (distancia, vecino))
    return distancias

# Ejemplo
grafo = {
    'A': {'B': 1, 'C': 4},
    'B': {'A': 1, 'C': 2, 'D': 5},
    'C': {'A': 4, 'B': 2, 'D': 1},
    'D': {'B': 5, 'C': 1}
}
print(dijkstra(grafo, 'A'))  # Salida: {'A': 0, 'B': 1, 'C': 3, 'D': 4}
```

5. Knapsack Problem

Descripción: Un problema de optimización que busca maximizar el valor total en una mochila sin exceder un peso máximo.

Ejemplo en Python

```
def knapsack(weights, values, capacity):
    n = len(weights)
    dp = [[0] * (capacity + 1) for _ in range(n + 1)]
    for i in range(n + 1):
        for w in range(capacity + 1):
```

```
            if i == 0 or w == 0:
                dp[i][w] = 0
            elif weights[i - 1] <= w:
                dp[i][w] = max(values[i - 1] + dp[i - 1][w - weights[i - 1]], dp[i - 1][w])
            else:
                dp[i][w] = dp[i - 1][w]
    return dp[n][capacity]

# Ejemplo
weights = [2, 3, 4, 5]
values = [3, 4, 5, 6]
capacity = 5
print(knapsack(weights, values, capacity))  # Salida: 7
```