PUCE Virtual

DOCUMENTO DE PROFUNDIZACIÓN

Estructura de Datos

Laboratorio de Contenido 2: Búsqueda en Anchura (BFS)

Laboratorio de Contenido 2: Búsqueda en Anchura (BFS)

Explicación básica

El algoritmo BFS es un método de búsqueda para grafos y árboles. Su estrategia consiste en explorar todos los vecinos de un nodo antes de pasar al siguiente nivel de profundidad.

Imagina que lanzas una piedra en un estanque: las ondas se expanden uniformemente desde el centro hacia afuera. BFS funciona igual, garantizando que, en grafos sin pesos, la primera vez que alcances un nodo, habrás llegado por el **camino más corto**.

Objetivos

- Comprender la estructura de datos tipo **Cola (FIFO)** y su rol en BFS.
- Implementar una búsqueda que encuentre la ruta mínima entre dos puntos.
- Analizar la complejidad temporal y espacial del algoritmo.

Caso práctico

Supongamos que trabajas en una red social. Quieres saber cuál es el "grado de separación" entre dos usuarios. Por ejemplo, ¿cuántos conocidos hay entre Ana y Juan? BFS es la herramienta perfecta para calcular esta distancia mínima de contactos.

Problema

Dado un mapa de conexiones (grafo), determinar si existe un camino entre un **Usuario A** (Inicio) y un **Usuario B** (Destino) y mostrar la ruta exacta.

Aplicación paso a paso

Para este laboratorio, utilizaremos un diccionario para representar el grafo y la clase deque para gestionar la cola de exploración.

Implementación en Python

Paso 1: Definir el Grafo

```
from collections import deque

# Red social: nombres y sus listas de amigos
red_social = {
```

```
    "Ana": ["Beto", "Carla"],
    "Beto": ["Ana", "David", "Elena"],
    "Carla": ["Ana", "Felipe"],
    "David": ["Beto"],
    "Elena": ["Beto", "Felipe"],
    "Felipe": ["Carla", "Elena"]
}
```

Paso 2: Implementar BFS para encontrar el camino

```
def buscar_camino_corto(grafo, inicio, destino):
    # La cola guarda tuplas: (nodo_actual, camino_recorrido)
    cola = deque([(inicio, [inicio])])
    visitados = {inicio}

    while cola:
        (nodo, camino) = cola.popleft()

        # Si encontramos el destino, retornamos el camino
        if nodo == destino:
            return camino

        # Explorar vecinos
        for vecino in grafo.get(nodo, []):
            if vecino not in visitados:
                visitados.add(vecino)
                # Creamos un nuevo camino sumando el vecino al actual
                cola.append((vecino, camino + [vecino]))

    return None # No hay conexión

# Ejecución
ruta = buscar_camino_corto(red_social, "Ana", "Felipe")
print(f"La ruta más corta es: {ruta}")
```

Conclusiones

- **Eficacia en Rutas Cortas:** BFS es óptimo para encontrar el camino mínimo en grafos donde todas las aristas tienen el mismo "costo".

- **Gestión de Memoria:** A diferencia de DFS (Búsqueda en Profundidad), BFS puede consumir mucha memoria en grafos con un factor de ramificación alto, ya que debe almacenar todos los nodos de un nivel en la cola.
- **Uso de la Cola:** La propiedad **FIFO** (First-In, First-Out) es lo que garantiza que exploremos por niveles.