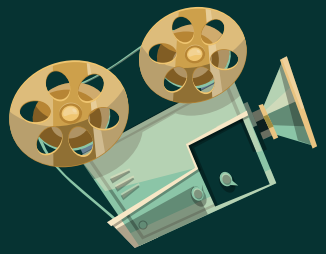
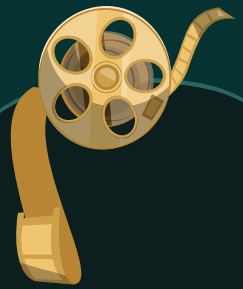


PROBLEMA 1 GRADOS DE SEPARACIÓN



Actores

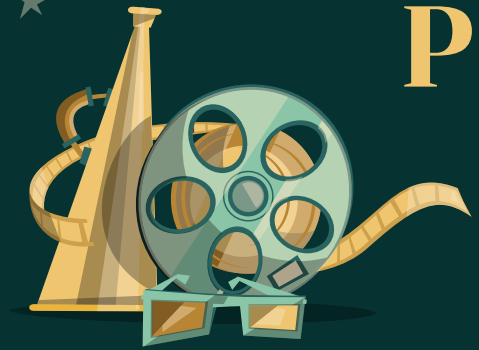
Inteligencia Artificial

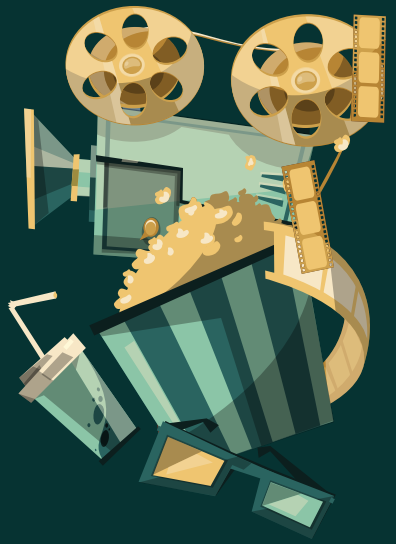
Introducción al Proyecto

En este problema, nos interesa encontrar el camino más corto entre dos actores cua-alquiera eligiendo una secuencia de películas que los conecte

I.

PROBLEMA VS SOLUCIÓN





Algoritmo de Búsqueda en Amplitud (BFS)

BFS es un algoritmo de búsqueda utilizado para explorar o recorrer estructuras de datos como grafos y árboles. Comienza desde un nodo raíz y explora todos sus nodos vecinos antes de moverse a los nodos vecinos de sus vecinos.

Dentro del Codigo

```python
if metodo == 'bfs':
    # Inicializamos una cola para BFS
    queue = deque()
    # Creamos un set para llevar un registro de las personas visitadas
    visited = set()
    # Inicializamos la cola con la persona de origen y un camino vacío
    queue.append((source, []))

    while queue:
        person, path = queue.popleft()
        # Marcamos la persona actual como visitada
        visited.add(person)

        if person == target:
            return path  #Exito

        # Exploramos a los vecinos (personas que compartieron películas)
        neighbors = neighbors_for_person(person)
        for movie_id, neighbor in neighbors:
            if neighbor not in visited:
                # Creamos un nuevo camino extendiendo el camino actual
                new_path = path + [(movie_id, neighbor)]
                queue.append((neighbor, new_path))

    #No exito
    return None
```

Algoritmo de Búsqueda en Profundidad (DFS)

DFS es un algoritmo de búsqueda utilizado para explorar o recorrer estructuras de datos como grafos y árboles. Comienza desde un nodo raíz y continúa explorando hacia abajo en un camino tanto como sea posible antes de retroceder y explorar otros caminos.

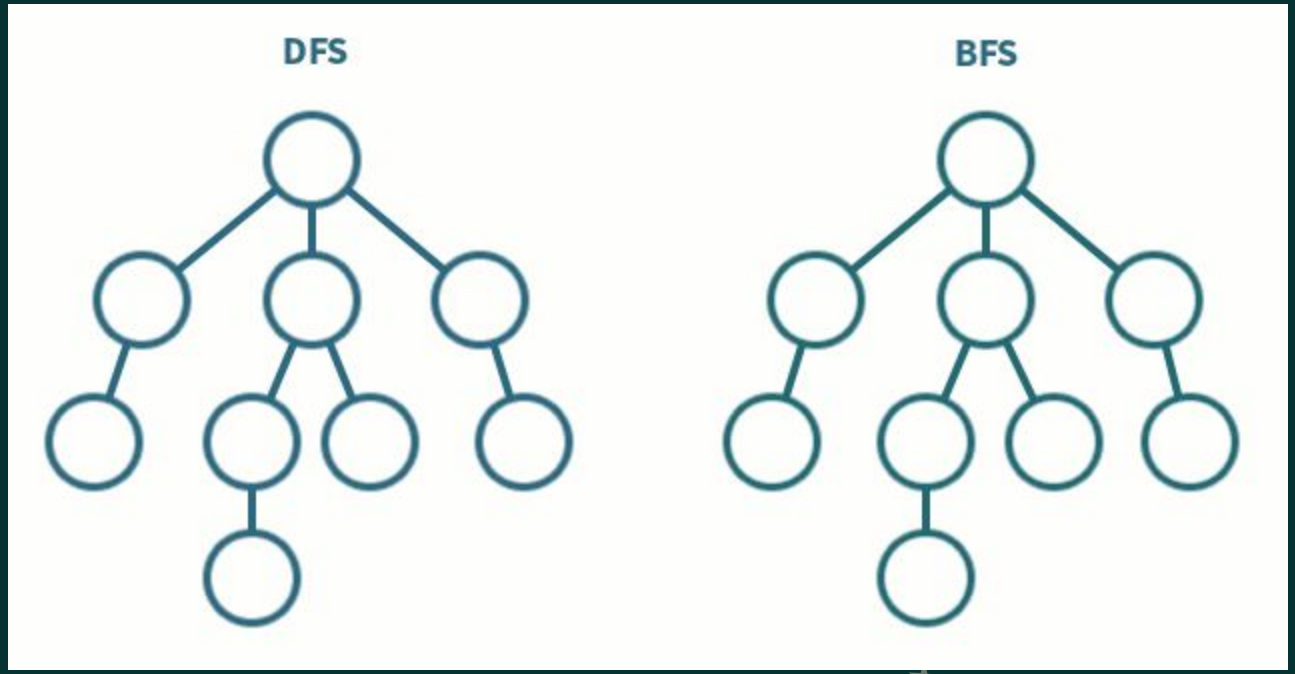
Dentro del Codigo

```python
if metodo == 'dfs':
    # Pila para realizar una búsqueda en profundidad (DFS)
    stack = []
    # Conjunto para llevar un registro de las personas visitadas
    visited = set()
    # Inicializamos la pila con la persona de origen y un camino vacío
    stack.append((source, []))

    while stack:
        person, path = stack.pop()
        # Marcamos la persona actual como visitada
        visited.add(person)

        # Verificamos si la persona actual es el objetivo
        if person == target:
            return path  # Devolvemos el camino desde la fuente hasta el objetivo

        # Exploramos a los vecinos (personas que compartieron películas)
        neighbors = neighbors_for_person(person)
        for movie_id, neighbor in neighbors:
            if neighbor not in visited:
                # Creamos un nuevo camino extendiendo el camino actual
                new_path = path + [(movie_id, neighbor)]
                stack.append((neighbor, new_path))

    # Si no se encuentra un camino, devolvemos None
    return None
```

II.

BFS VS DFS

¿Cual es mejor para esta situación?


DFS
BFS

BFS vs DFS

VENTAJAS

- Encuentra la ruta más corta en términos de la cantidad de películas (grados de separación) entre los actores.
- Garantiza encontrar la ruta más corta en grafos no ponderados y finitos.

DESVENTAJAS

- Puede requerir más memoria que DFS, ya que mantiene una cola de nodos por explorar.
- Puede ser más lento en comparación con DFS en grafos profundos.

VENTAJAS

- Puede ser más eficiente en términos de memoria en grafos profundos, ya que utiliza una pila o recursión.
- Puede encontrar una ruta en un grafo más rápidamente si está cerca del objetivo.

DESVENTAJAS

- No garantiza encontrar la ruta más corta en términos de la cantidad de películas.
- Puede quedarse atrapado en ciclos infinitos en grafos con ciclos.

PRUEBA PRÁCTICA

Ejemplo

CONCLUSION

Entonces, ¿cuál es mejor? Depende de tus objetivos y del problema específico:

Si deseas encontrar la ruta más corta (en términos de películas) entre dos actores y la estructura de tu grafo es un grafo no ponderado, BFS es una mejor opción.