PROBLEMA DE LOS ACTORES

Problema 1

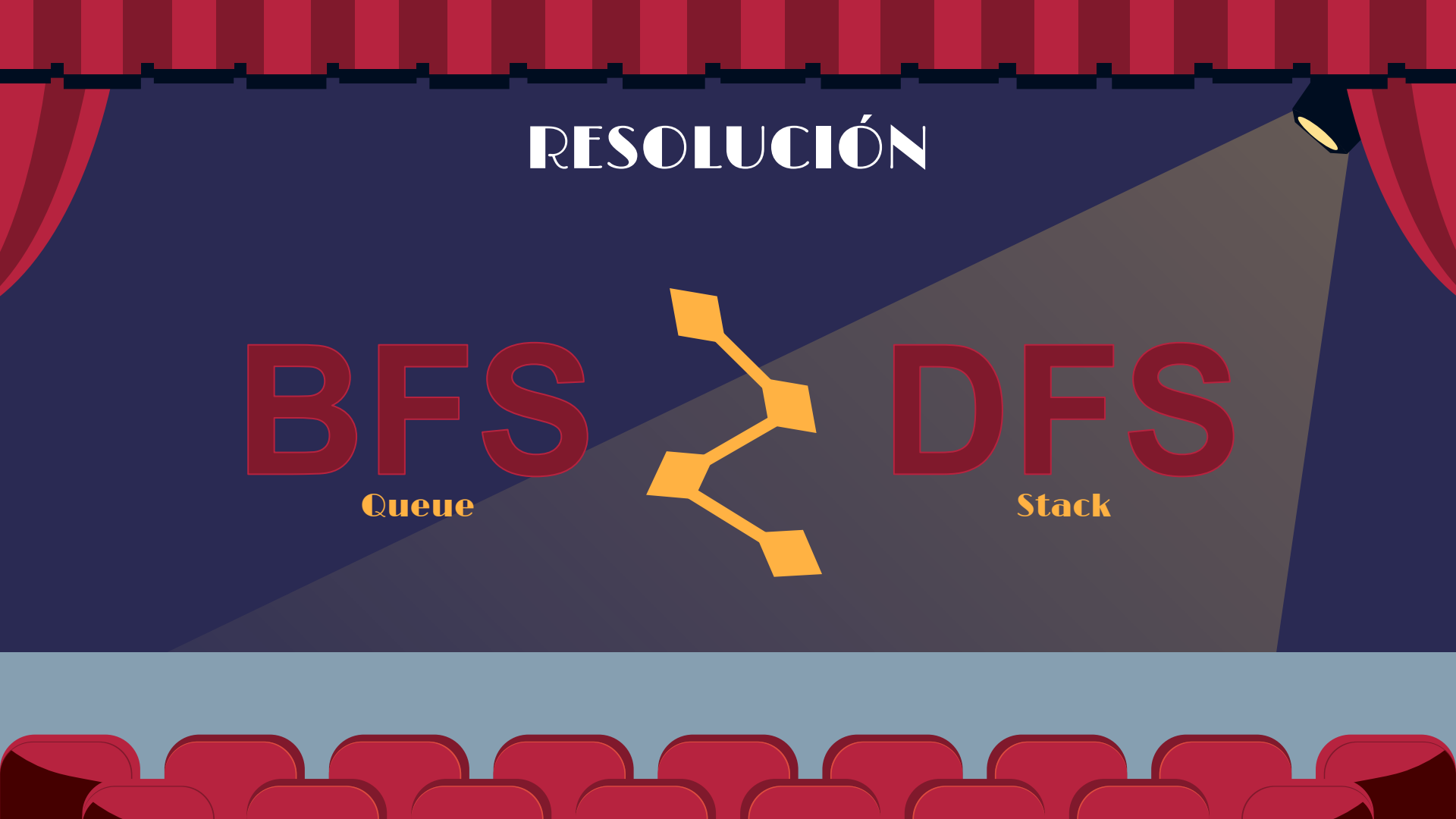
OBJETIVO

¿

?

OBJETIVO

RESOLUCIÓN



```python
def shortest_path(actorOrigen, actorDestino):
    """
    Returns the shortest list of (movie_id, person_id) pairs
    that connect the source to the target.

    If no possible path, returns None.

    """
    #State = ID del Actor
    #Parent = Nodo Padre
    #Action = ID de la Pelicula que asoicia al actor del nodo padre con el actor del nodo actual
    #Node = (State, Parent, Action)

    #Inicializamos Frontera
    frontera = QueueFrontier()

    #Inicializamos Conjunto de Nodos Explorados
    explorados = set()

    #Inicializamos Nodo de Inicio
    nodoInicial = Node(state=actorOrigen, parent=None, action=None)

    #Agregamos Nodo Inicial a la Frontera
    frontera.add(nodoInicial)
```

```python
#Loop infinito hasta encontrar la solución o recorrer todos los nodos
while True:

    #Si la frontera esta vacia, no hay solución
    if frontera.empty():
        return None

    #Sacamos el primer nodo de la frontera para analizarlo
    #Inicialmente será el nodoInicial (actorOrigen)
    node = frontera.remove()

    #Si el actor del nodo es el actorDestino, encontramos la solución
    if node.state == actorDestino:
        #Creamos una lista de tuplas (movie_id, person_id) que representan la solución
        solution = []

        #Recorremos los nodos hasta llegar al nodoInicial (actorOrigen)
        while node.parent is not None:
            #Agregamos la tupla (movie_id, person_id) a la solución (path)
            solution.append((node.action,node.state))
            #El nodo actual pasa a ser el nodo padre (Vamos haciendo backtracking)
            node = node.parent

        #Invertimos la solución para que quede en el orden correcto (actorOrigen -> actorDestino) y no (actorDestino -> actorOrigen)
        solution.reverse()
        #Devolvemos la solución
        return solution

    #Si el actor del nodo no es el actorDestino, lo agregamos al conjunto de nodos explorados para no reexplorarlo
    explorados.add(node.state)

    #Agregamos los vecinos del nodo a la frontera
    #Por cada pelicula en la que participa el actor del nodo, agregamos a la frontera los actores que participan en esa pelicula
    for movie_id, person_id in neighbors_for_person(node.state):
        #Se agregan solo si ya no estan en la frontera y no fueron explorados
        if not frontera.contains_state(person_id) and person_id not in explorados:
            #El nodo actual se convierte en el padre, la pelicula en la accion con la que se llega al actor del nodo
            child = Node(state=person_id, parent=node, action=movie_id)
            #Agrego a la frontera
            frontera.add(child)
```

PROCESO

Paso 1

Nodo Inicial

Agregar nodo inicial (actor origen) a la frontera

Paso 2

Loop Solución

Ingresamos al loop hasta dar con la solución o encontrar que no hay solución

PROCESO

Paso 2.1

Frontera Vacía

Si la frontera ya está vacía quiere decir que no hay solución

Paso 2.2

Elección de Nodo

Sacamos un nodo de la frontera y comprobamos si es la solución

- Sí es la solución creamos el path (Paso 3)
- Si no es la solución, marcamos el nodo como explorado y buscamos los vecinos (Paso 4)

PROCESO

Paso 3.1

Construcción del Path

Mientras no se trate del nodo inicial, iremos agregando la película y el actor del nodo actual al arreglo de solución, luego cambiaremos el nodo actual a su nodo padre para ir subiendo hasta el nodo inicial.

Paso 3.2

Formato de la solución

Invertimos el arreglo de la solución debido a la regresión para tener la orientación correcta.

PROCESO

Paso 4.1

Nodo explorado

Agregamos el nodo actual a la lista de explorados

Paso 4.2

Identificación de vecinos

Identificamos todas las películas en las que actuó el actor y todos los actores con los que actuó en cada una. Si el actor con el que actuó no se encuentra en la frontera o en explorados lo agrego a la frontera.

Nodo Inicial

A1

Actor Origen

A2

Actor Destino

Nodo → A1 != A2

Frontera

A1

Explorados

Nodo Inicial

A1

Actor Origen

A2

Actor Destino

Nodo → A1 != A2

Frontera

Explorados

A1

Películas

M1
M2
M3

Actores

A3
A4
A5

Nodo Inicial

A1

Actor Origen

A2

Actor Destino

Nodo → A3 != A2

Frontera

A3
A4
A5

Explorados

A1

- Al usar QueueFrontier se elige el más antiguo (A3)
- Para StackFrontier se hubiese elegido (A5)

Nodo Inicial

A1

Actor Origen

A2

Actor Destino

Nodo → A3 != A2

Frontera

A4
A5

Explorados

A1
A3

Películas

M1
M5

Actores

A4
A5
A2
A1

Nodo Inicial

A1

Actor Origen

A2

Actor Destino

Nodo → A4 != A2

Frontera

A5
A2

Explorados

A1
A3
A4

Nodo Inicial

A1

Actor Origen

A2

Actor Destino

Nodo → A5 != A2

Frontera

A2
A6
A7
A11

Explorados

A1
A3
A4
A5

Nodo Inicial

A1

Actor Origen

A2

Actor Destino

Nodo → A2 == A2

Frontera

A6
A7
A11

Explorados

A1
A3
A4
A5
A2

Nodo Inicial

A1

Actor Origen

A2

Actor Destino

Nodo → A2 == A2

Solución

[]

[(A2,M5)]

[(A2,M5),(A3,M1)]

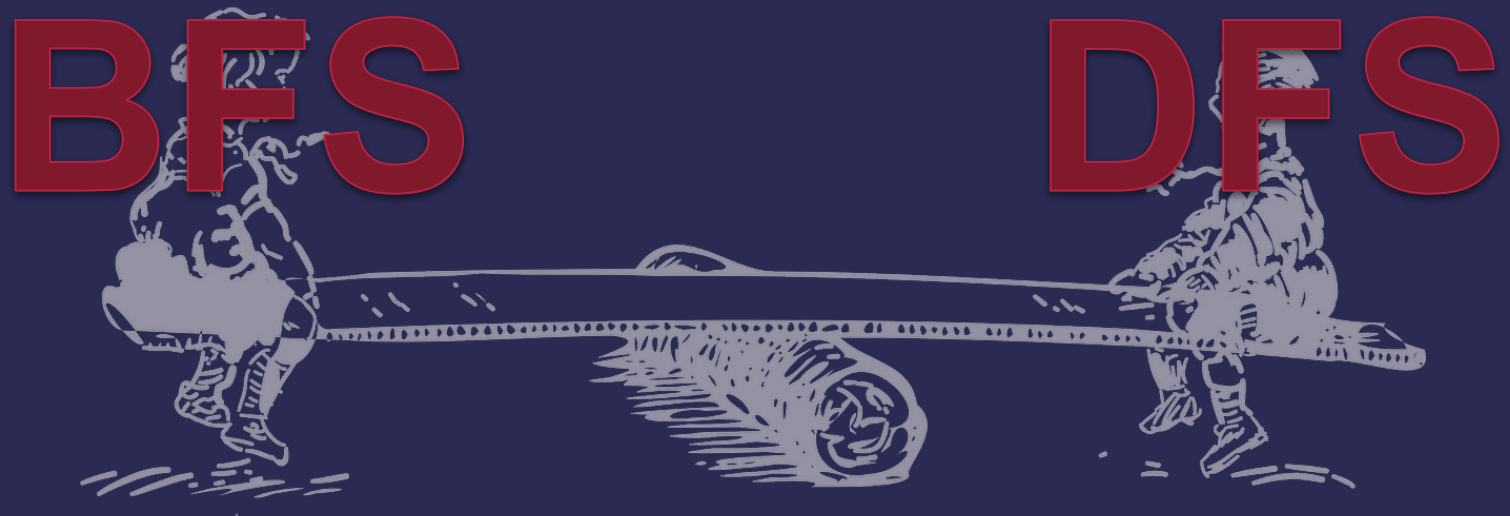
Nodo	State	Action	Padre
A2	A2	M5	A3
A3	A3	M1	A1
A1	A1	None	None

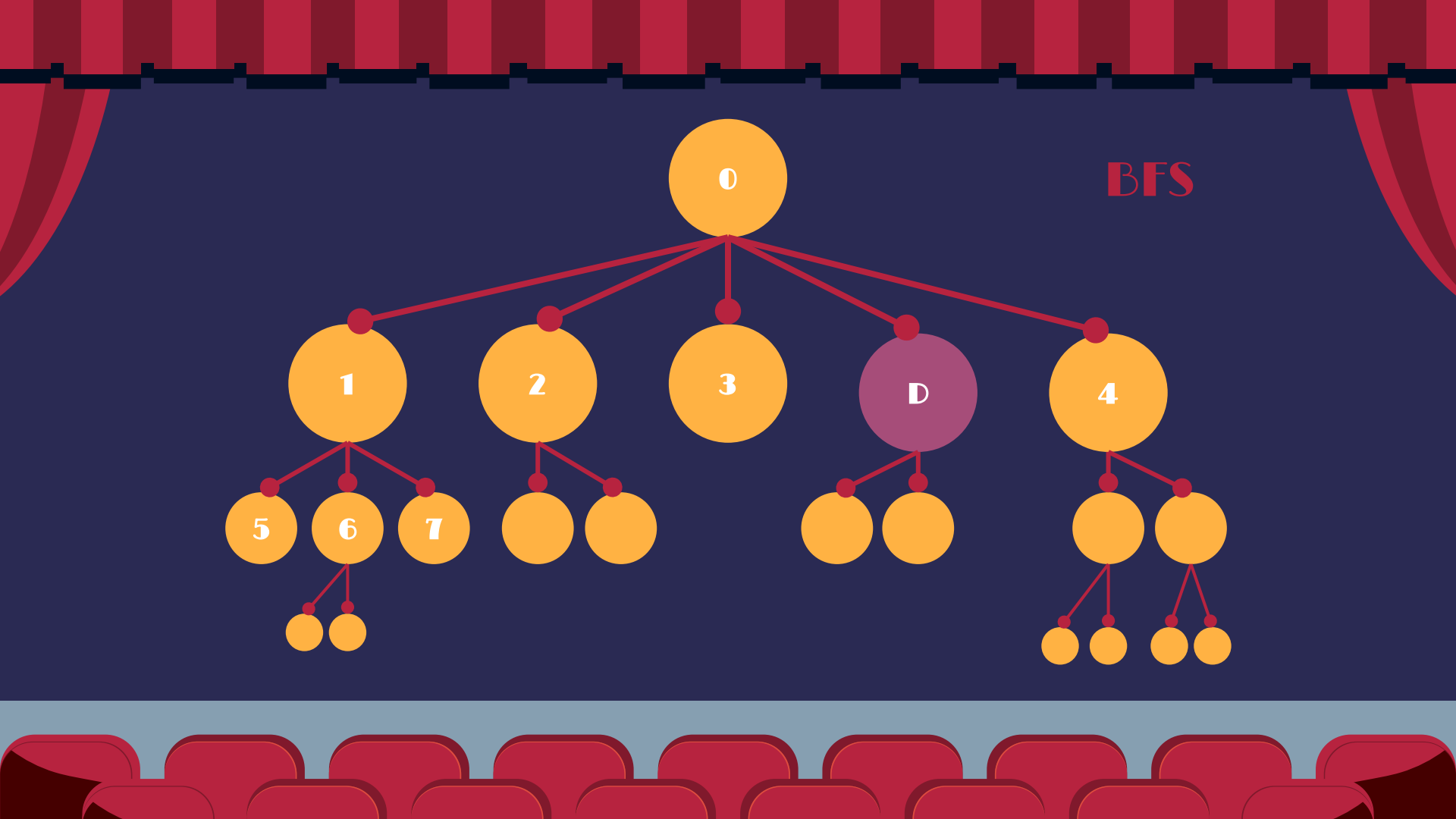
[(A2,M5) , (A3,M1)]

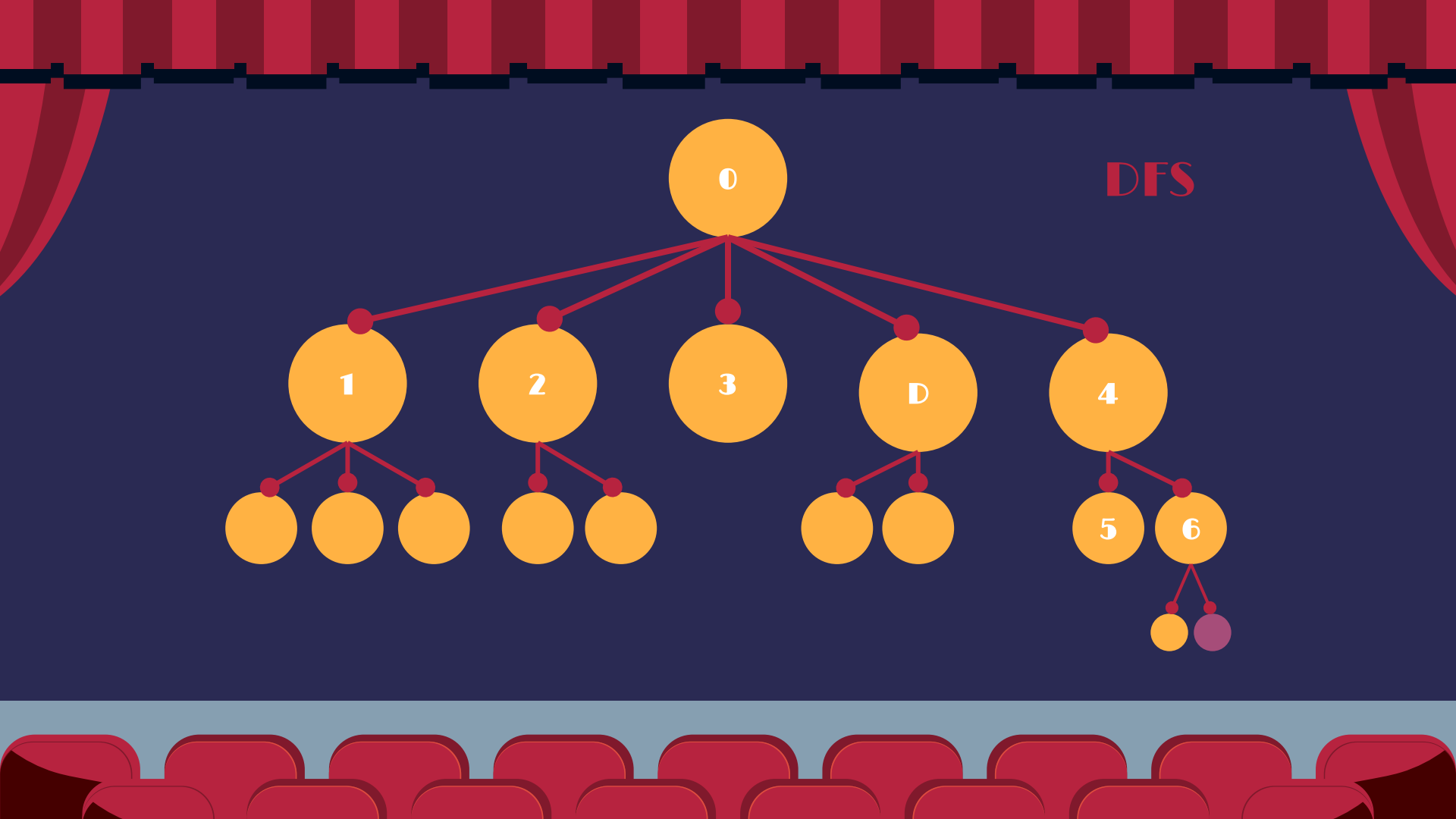
⇌

[(A3,M1) , (A2,M5)]

RENDIMIENTO




BFS
0
1
2
3
D
4
5
6
7


DFS
0
1
2
3
D
4
5
6

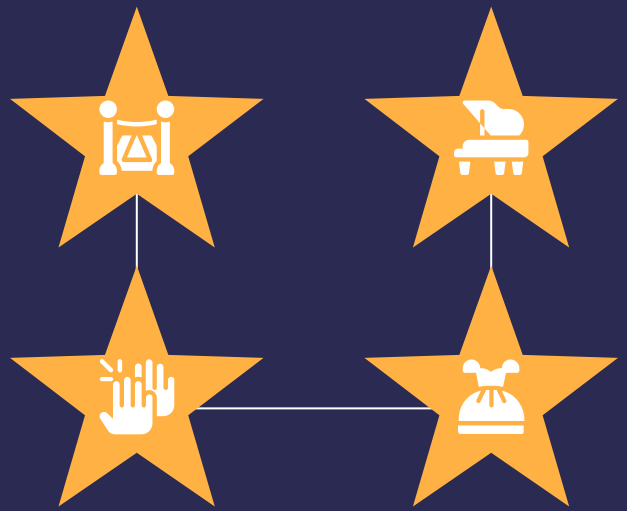
MEJORAS

Profundidad

Establecer un límite en la profundidad que se desea abarcar

Paralelizar

Asignar distintos nodos a distintos hilos



Bidireccionalidad

Comenzar a buscar desde el origen y el destino hasta que se encuentren

Función Heurística

Elegir otro algoritmo más complejo, como Dijkstra o A* y estimar costo

IMPLEMENTACIÓN

¡GRACIAS!

2

1

3