

Grado en Ingeniería Informática
Departamento de Ingeniería Informática
Universidad de Cádiz

Tema 4

Agentes basados en objetivos
Formalización y Búsqueda No Informada

elisa.guerrero@uca.es

Tema 4. Agentes basados en objetivos

Objetivos

Al finalizar el tema 4 el alumno ha de ser capaz de:

1. Identificar los tipos de problemas que pueden ser resueltos mediante estrategias de búsqueda en espacios de estados.
2. Realizar la formalización de problemas de búsqueda utilizando la información específica del dominio del problema.
3. Implementar en un lenguaje de programación dichas formalizaciones.
4. Comprender el funcionamiento de las estrategias de búsqueda a ciegas.
5. Resolver los problemas planteados utilizando las distintas técnicas de búsqueda no informada.
6. Analizar el coste de las distintas técnicas para evaluar las ventajas y desventajas de cada método.
7. Implementar los distintos algoritmos estudiados.

Tema 4. Agentes basados en objetivos

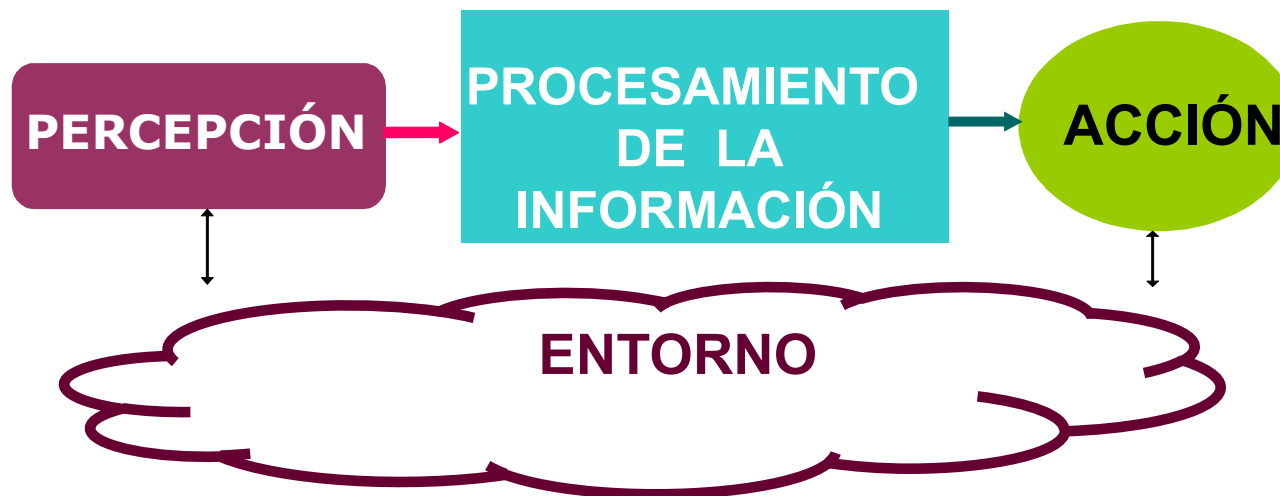
Índice del Tema

1. Introducción a los agentes basados en objetivos
2. Formalización de problemas de búsqueda
3. Estrategias de búsqueda
 1. Algoritmo genérico
 2. Búsqueda en Anchura
 3. Búsqueda en Profundidad
 4. Análisis del rendimiento

1. Introducción

Agente Inteligente

- Un **agente inteligente** es aquél que emprende la mejor acción posible ante una situación dada.
(Russell & Norvig, 2004)



1. Introducción

Agente basado en objetivos: Agente que Resuelve Problemas

- **Objetivo:** definido en función de la situación final deseable y mediante un conjunto de estados del mundo (los que satisfacen el objetivo).
- **Formalización del Problema:** dado un objetivo, el proceso de especificación de las acciones y estados que se van a considerar.
- **Tarea del Agente:** encontrar la secuencia de acciones que permiten obtener un estado objetivo (Búsqueda en un espacio de estados).

1. Introducción

Ejemplos de problemas



5	4	
6	1	8
7	3	2

Start State

1	2	3
8		4
7	6	5

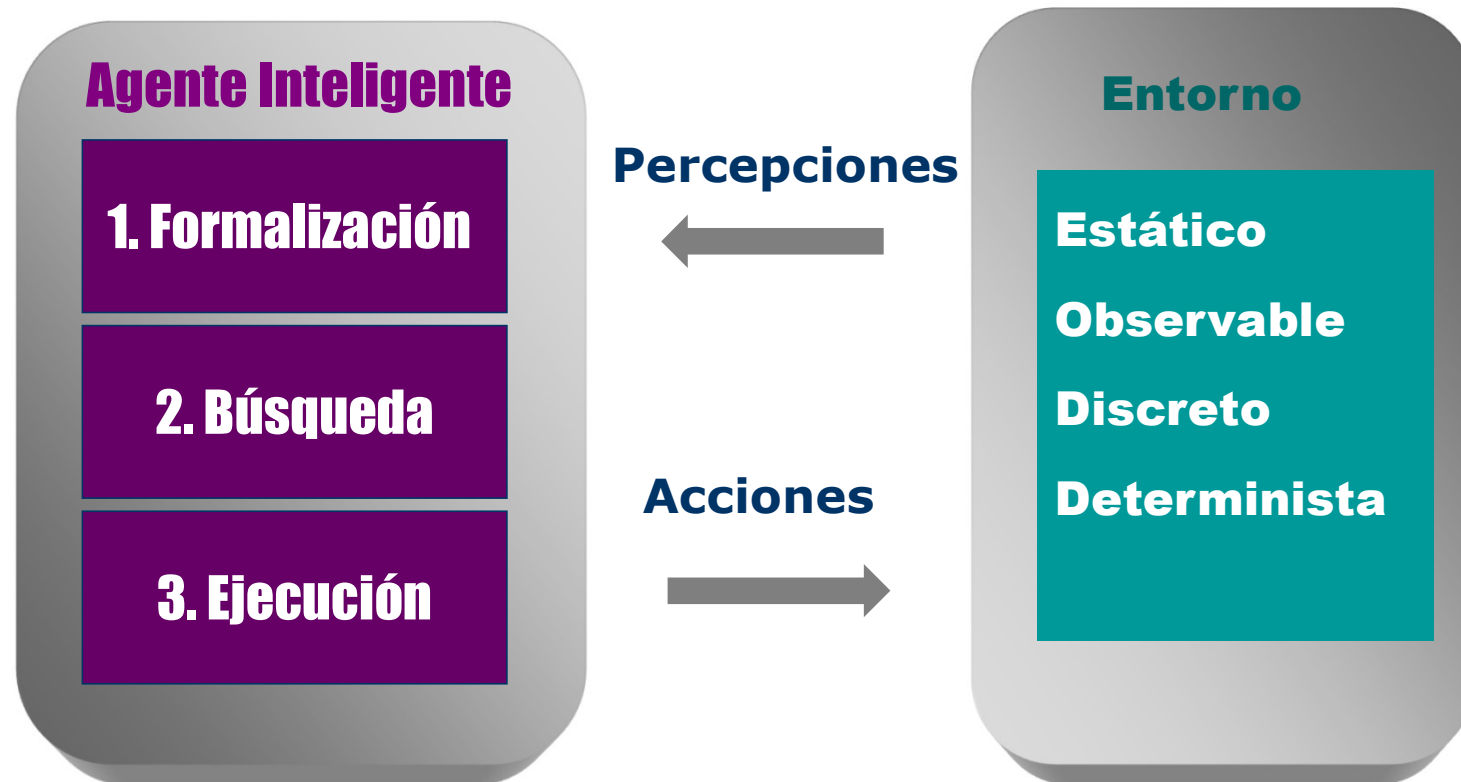
Goal State



0				
1				
2				
3				SALIDA
	0	1	2	3

1. Introducción

Fases para Agentes basados en Objetivos



1. Introducción

Condiciones del Entorno de resolución

ENTORNO

- **Estático:** no se tienen en cuenta los cambios que puedan ocurrir a posteriori
- **Observable:** permite definir el **estado inicial**
- **Discreto:** permite enumerar las **líneas de acción**
- **Determinista**, ya que el **siguiente estado** está totalmente determinado por el estado actual y la acción posible a tomar

2. Formalización

Un problema se define mediante:

- **Estado Inicial**
- **Test Objetivo**
- **Lista de operadores (acciones)**
- **Sucesores (espacio de estados)**
- **Camino o solución**
- **Coste**

2. Formalización

- **Estado Inicial:** situación o configuración inicial desde la que se inicia la resolución del problema.

5	4	
6	1	8
7	3	2

- **Test Objetivo:** verifica si un estado dado es un estado objetivo, una solución al problema.
¿El estado S es igual al estado final requerido?



1	2	3
8		4
6	7	5

1	2	3
8		4
7	6	5

2. Formalización

- **Lista de Operadores:** ACCIONES que generan nuevos estados **Sucesores**.
- **Funciones:** Se ha de pasar de un estado a otro de acuerdo a una serie de acciones u operaciones bien definidas y que cumplan una serie de reglas.
 - esValido (s)
 - aplicaOperador(o, s)
- **Camino:** secuencia de estados conectados por una secuencia de acciones.
- **Coste de la Solución:** medida de rendimiento → Solución óptima

2. Formalización

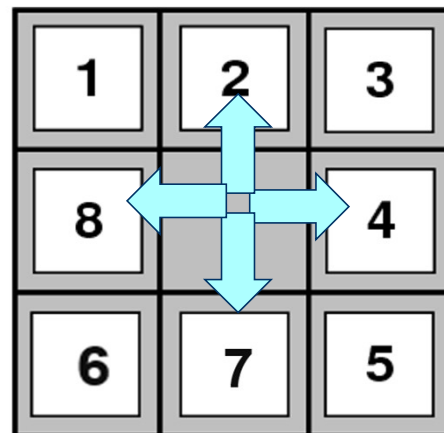
Estado Actual, Sucesores y Operadores

- Los **sucesores** son los nuevos estados válidos que se generan a partir de una determinada situación o estado del problema (el **estado actual**).
- Los operadores son las acciones u operaciones bien definidas mediante reglas que permiten pasar de un estado a otro estado sucesor. Se necesitan dos funciones para generar los sucesores:
 - **esValido**: función que comprueba que se cumplen las reglas para poder realizar un movimiento o una acción que lleve a un nuevo estado válido. Devuelve Verdadero o Falso.
 - **aplicaOperador**: función que realiza una acción aprobada previamente con esValido, debe generar un nuevo estado (este estado debe ser un estado Válido por tanto).

2. Formalización

Problema del 8-puzzle

- Lista de Operadores: Intercambiar el hueco con una ficha **adyacente**:



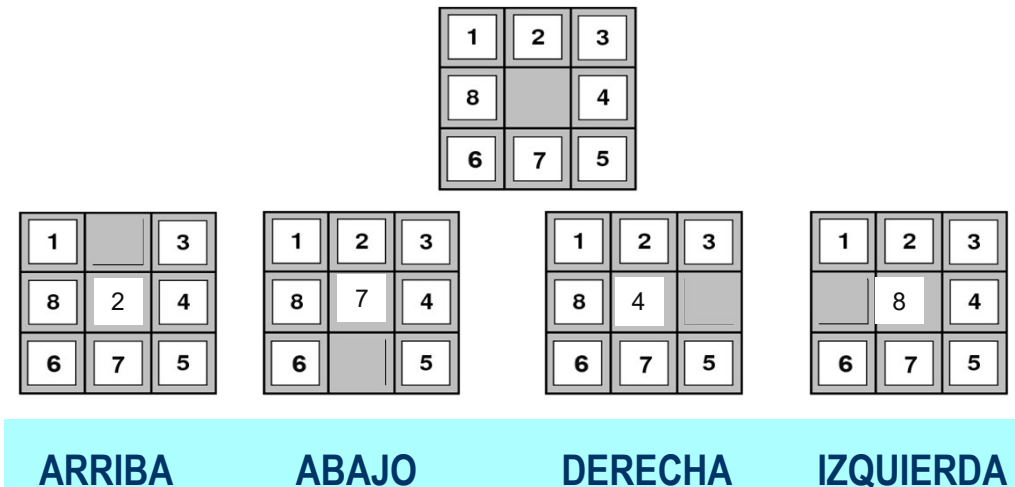
DERECHA
IZQUIERDA
ARRIBA
ABAJO

- **Solución:** Movimientos para llegar desde estado inicial al estado objetivo.
- **Coste del camino:** suma del coste de cada movimiento de que consta la solución.

2. Formalización

Problema del 8-puzzle

- **Sucesores:** nuevos estados generados a partir del actual, se usan las dos funciones:
 - **esValido:** determinar si hay una ficha adyacente a la ficha vacía para poder realizar el intercambio Arriba, Abajo, a la Izquierda o a la Derecha, cada movimiento tiene unas reglas específicas.
 - **aplicaOperador:** intercambiar la ficha vacía con otra ficha que se encuentre en una posición adyacente válida, aplicando uno de los 4 operadores.



2. Formalización 8-puzzle

Inicializaciones

```
import numpy as np
puzle_inicial = np.array([[5, 4, 0], [6, 1, 8], [7, 3, 2]])
```

5	4	
6	1	8
7	3	2

```
puzle_final = np.array([[1, 2, 3], [8, 0, 4], [7, 6, 5]])
```

1	2	3
8		4
6	7	5

```
movimientos = { "8": "ARRIBA",
                 "2": "ABAJO",
                 "4": "IZQUIERDA",
                 "6": "DERECHA" }
```

2. Formalización 8-puzzle

tEstado

```
from dataclasses import dataclass
```

```
import copy
```

```
@dataclass class tEstado:
```

```
.....
```

```
def __init__(self, ....):
```

```
.....
```

```
def __repr__(self) -> str:
```

```
    return .....
```

```
def crearHash(self) -> str:
```

```
    return .....
```

Mantendremos siempre una clase tEstado, pero su contenido cambiará de acuerdo al problema concreto

Así podremos reutilizar el código de búsqueda con independencia del problema

2. Formalización 8-puzzle tEstado

```
from dataclasses import dataclass
```

```
import copy
```

```
@dataclass class tEstado:
```

```
    tablero: np.ndarray
```

```
    fila: int
```

```
    col: int
```

```
    def __init__(self, tablero):
```

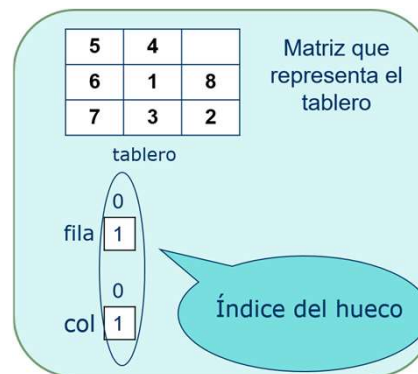
```
        .....
```

```
    def __repr__(self) -> str:
```

```
        return f'{self.tablero}\n Fila Hueco: {self.fila}\n Col Hueco: {self.col}\n'
```

```
    def crearHash(self) -> str:
```

```
        return f'{self.tablero.tobytes()}{self.fila}{self.col}'
```



Mantendremos siempre una clase tEstado, pero su contenido cambiará de acuerdo al problema concreto

Así podremos reutilizar el código de búsqueda con independencia del problema

2. Formalización 8-puzzle esValido

```
def esValido(move, estado):
```

```
    valido = False
```

```
    match movimientos[move]:
```

```
        case "ARRIBA":
```

```
            .....
```

```
        case "ABAJO":
```

```
            .....
```

```
        case "IZQUIERDA":
```

```
            .....
```

```
        case "DERECHA":
```

```
            .....
```

```
    return valido
```

El movimiento será válido siempre y cuando se respeten los límites del tablero.

2. Formalización 8-puzzle aplicaOp

```
def aplicaOperador(move, estado):
```

```
    nuevo = copy.deepcopy(estado)
```

```
    ficha = 0
```

```
    match movimientos[move]:
```

```
        case "ARRIBA":
```

```
            .....
```

```
        (...)
```

```
        case "DERECHA":
```

```
            .....
```

```
    .....
```

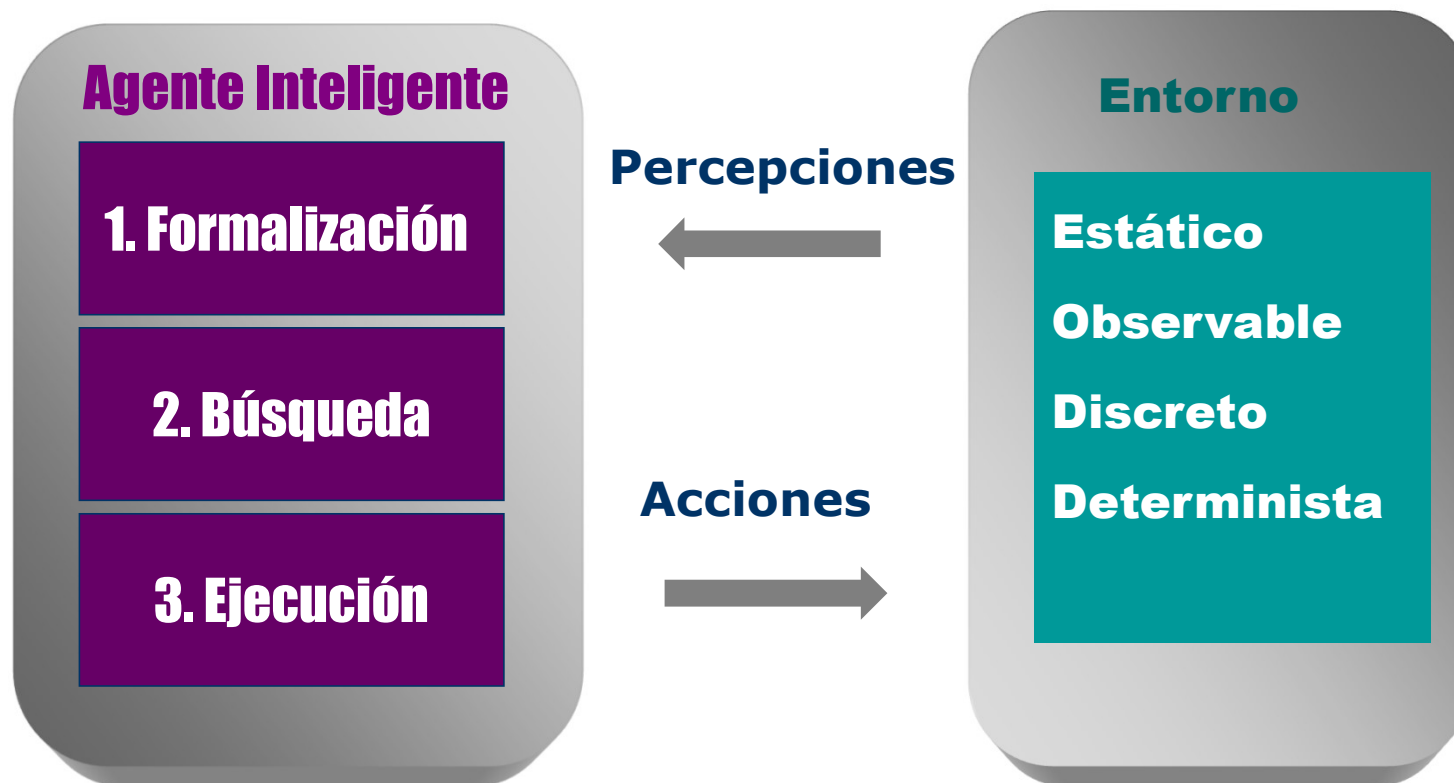
```
    return nuevo
```

Actualiza la posición del hueco dependiendo del movimiento

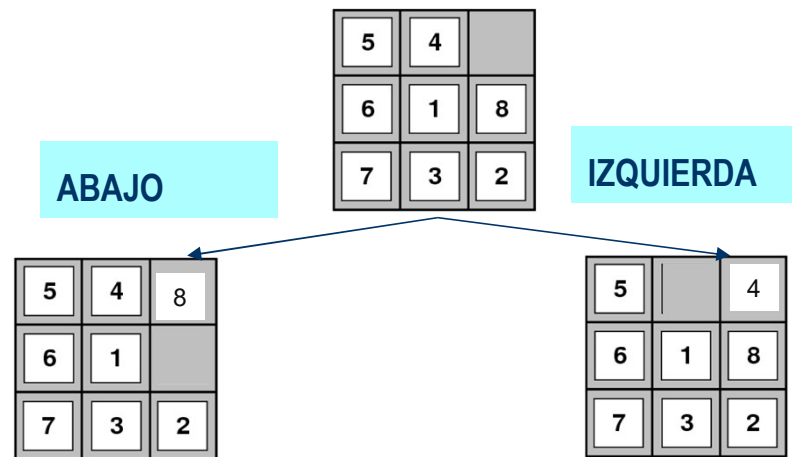
Intercambio del hueco por la posición de la ficha

2. Búsqueda

- **Búsqueda:** hallar la secuencia de acciones que conduzcan a un agente a un estado objetivo.



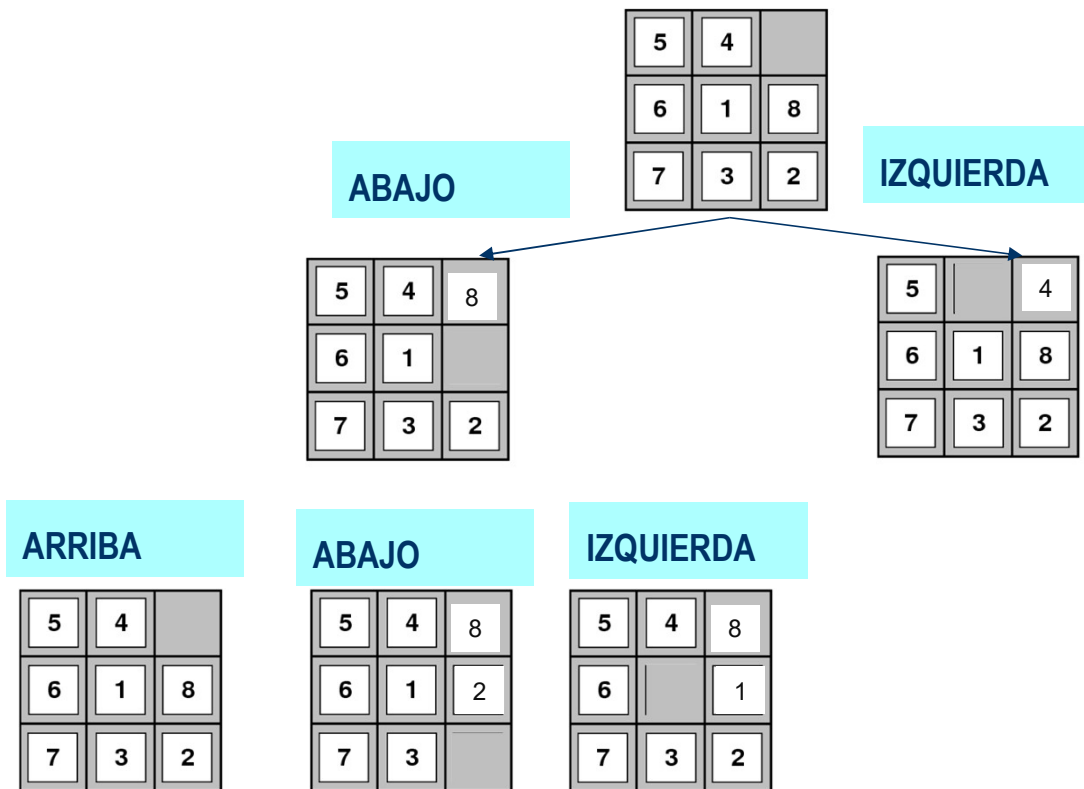
2. Búsqueda



movimientos = { "8": "ARRIBA",
"2": "ABAJO",
"4": "IZQUIERDA",
"6": "DERECHA" }

2. Búsqueda

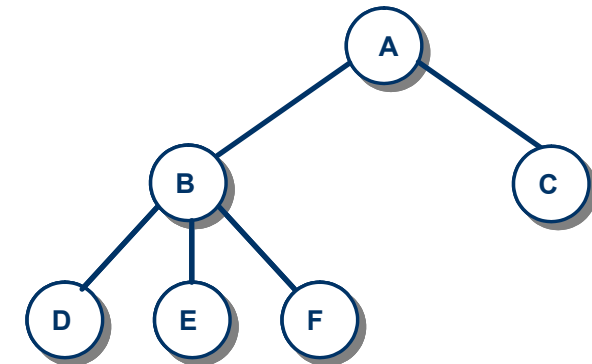
movimientos = { "8": "ARRIBA",
"2": "ABAJO",
"4": "IZQUIERDA",
"6": "DERECHA" }



2. Búsqueda

Diseño del Árbol o Grafo de Búsqueda

- **Estado**: configuración del mundo en un momento dado
- **Nodo**: estructura de datos para representar toda la información referente a cada estado:
 - Estado
 - Nodo Padre
 - Acción
 - Coste
 - Profundidad
- Selección: **Estrategia concreta de selección del nodo**
- ¿Es Objetivo el Nodo Actual?
- Función sucesor al nodo actual cuando no es un nodo objetivo
- Lista de Nodos **ABIERTOS**
- Lista de Nodos **CERRADOS**



2. Búsqueda

Funcionamiento básico de la búsqueda

Abiertos=Inicial
Mientras No_Objetivo(Actual) & NoVacía(Abiertos)
 Actual = Primero(Abiertos)
 Sucesores = **Expandir**(Actual)
 Abiertos = Abiertos + Sucesores
Fin_Mientras

2. Búsqueda

Funcionamiento básico de la búsqueda

```
Abiertos=Inicial
Mientras No_Objetivo(Actual) & NoVacía(Abiertos)
    Actual = Primero(Abiertos)
    Sucesores = Expandir(Actual)
    Abiertos = Abiertos + Sucesores
Fin_Mientras
```

```
función Expandir(tNodo: actual)
    desde op←1 hasta NUM_OPERADORES hacer
        si esValido(op,actual) entonces
            nuevo ← aplicaOperador(op,actual)
            Sucesores← {Sucesores + nuevo}
        fin_si
    fin_desde
    devolver Sucesores
```

2. Búsqueda

Tipos de búsqueda en un espacio de estados

Cuando hay varias posibilidades en el espacio de búsqueda, la estrategia debe determinar cuál es el siguiente estado a considerar

- **Búsqueda No Informada:** Exploración sistemática del espacio de búsqueda, pero sin información que ayude a determinar qué camino seguir
- **Búsqueda Informada o Heurística:** Se evalúa en cada momento qué estado pudiera ser mejor que otro para su expansión, utilizando cierta información en el dominio del problema

2. Búsqueda

Medidas del Rendimiento

Completa: la estrategia siempre que exista, encontrará una solución

Óptima: la estrategia siempre que exista solución, encontrará primero la mejor solución

Complejidad en tiempo: número de nodos generados durante la búsqueda

Complejidad en espacio: máximo número de nodos en memoria

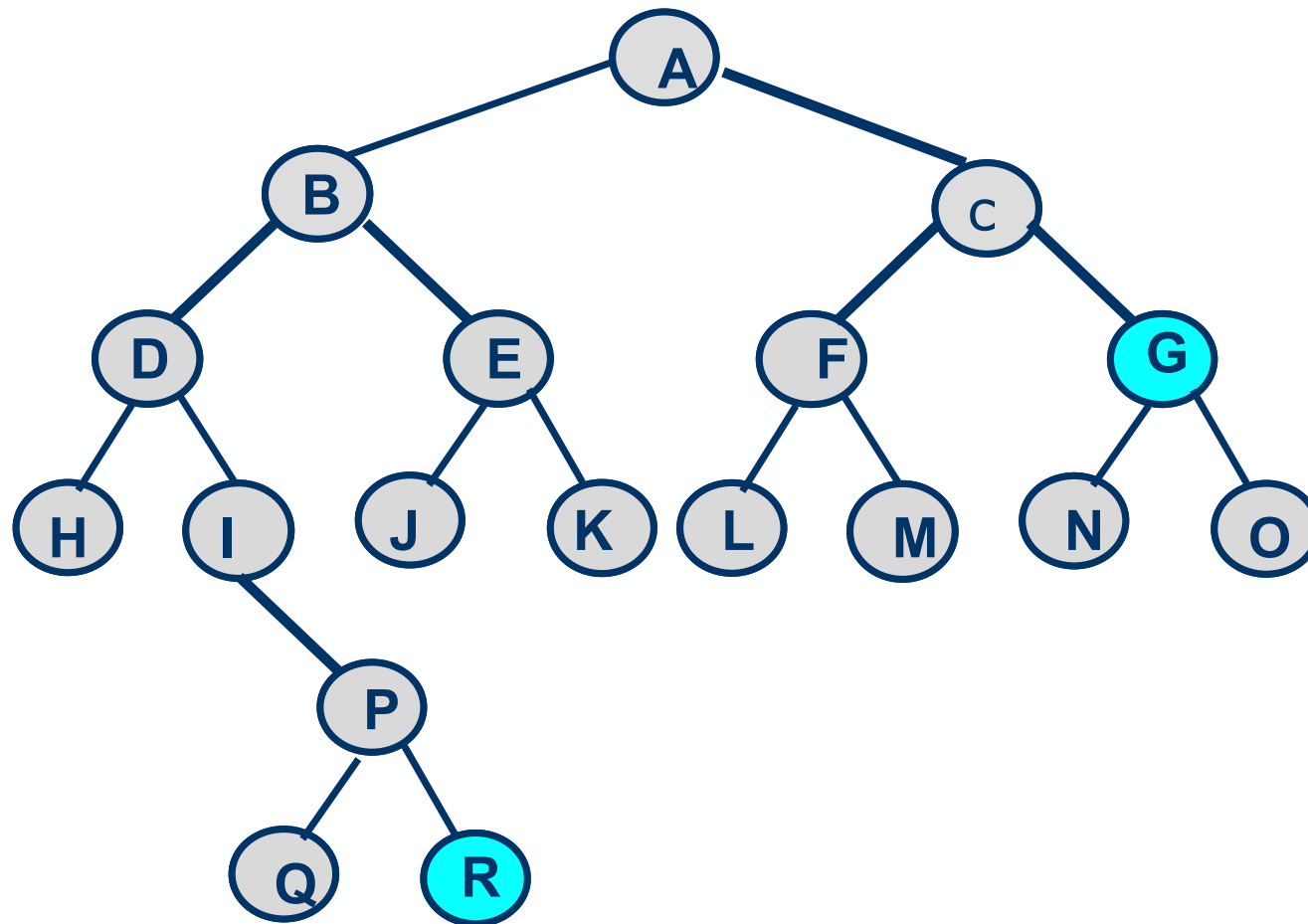
3. Estrategias de búsqueda no informada

- Búsqueda en Anchura
- Búsqueda en Profundidad
 - Con retroceso
 - Profundidad Limitada
 - Profundidad iterativa
- Búsqueda Biridireccional

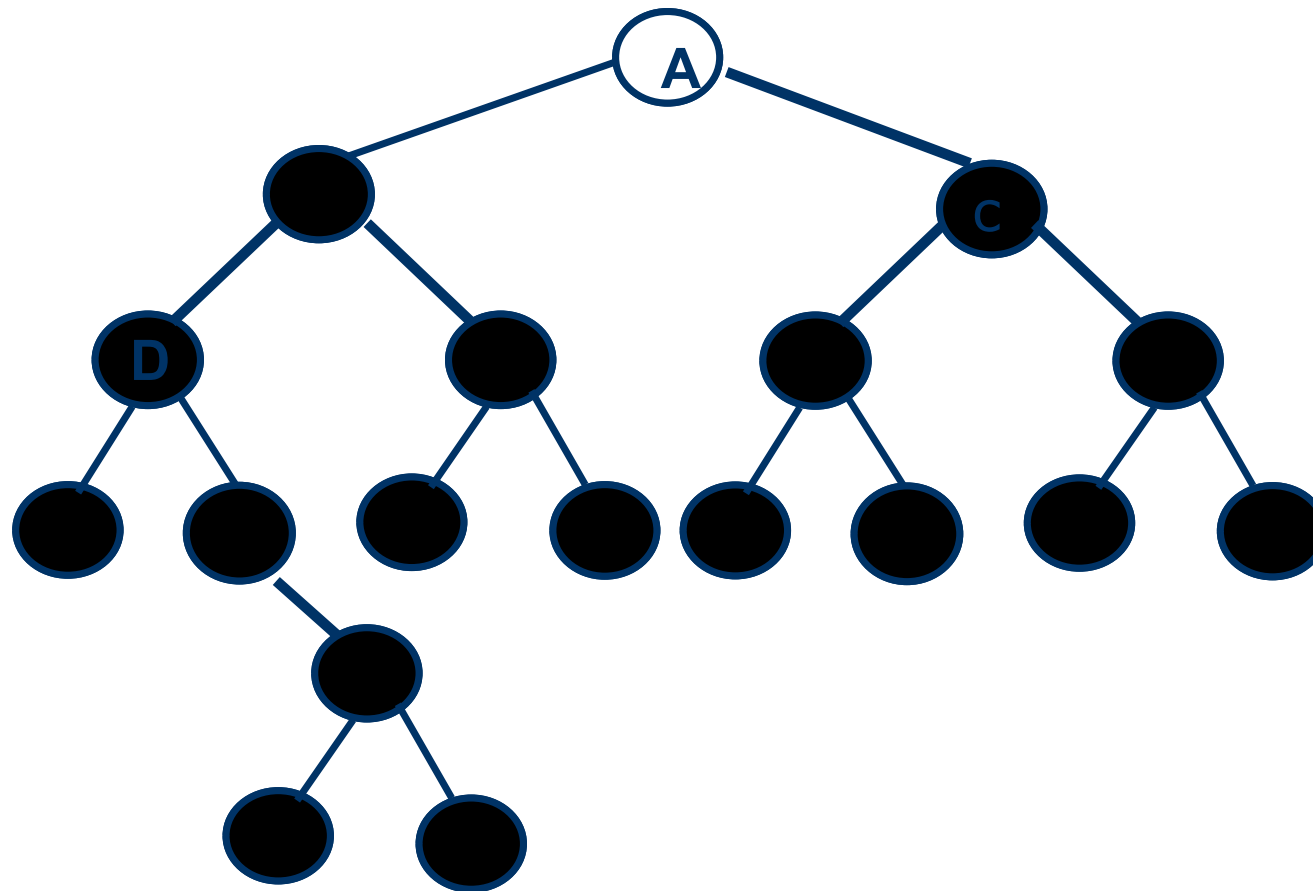
3.1 Búsqueda en Anchura

- Estrategia de selección del nodo actual de acuerdo a una estructura FIFO: el primer nodo que se genera es el primero que se expande (el más antiguo en la lista de Abiertos).
- Desde la perspectiva de la creación de un árbol de búsqueda: todos los nodos de un nivel se expanden antes que los del nivel siguiente.
- Estrategia **FIFO** = seleccionar el nodo más antiguo en la lista:
 - Seleccionar el nodo raíz (nivel 0)
 - Seleccionar los nodos generados a partir del nodo raíz (nivel 1)
 - Después los sucesores (nivel 2) y así sucesivamente

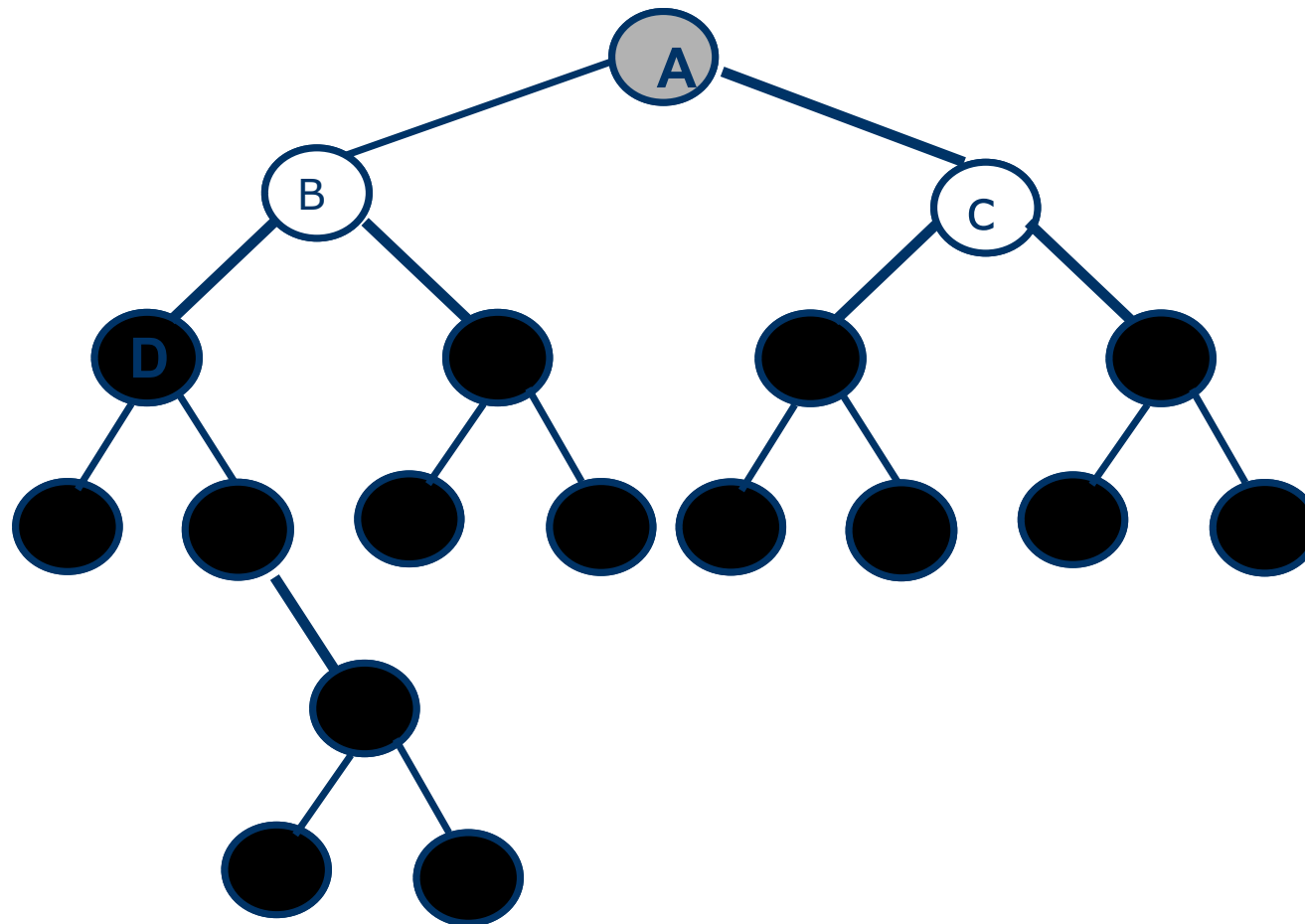
3.1 Búsqueda en Anchura.



3.1 Búsqueda en Anchura. Ejemplo Genérico



3.1 Búsqueda en Anchura. Ejemplo Genérico

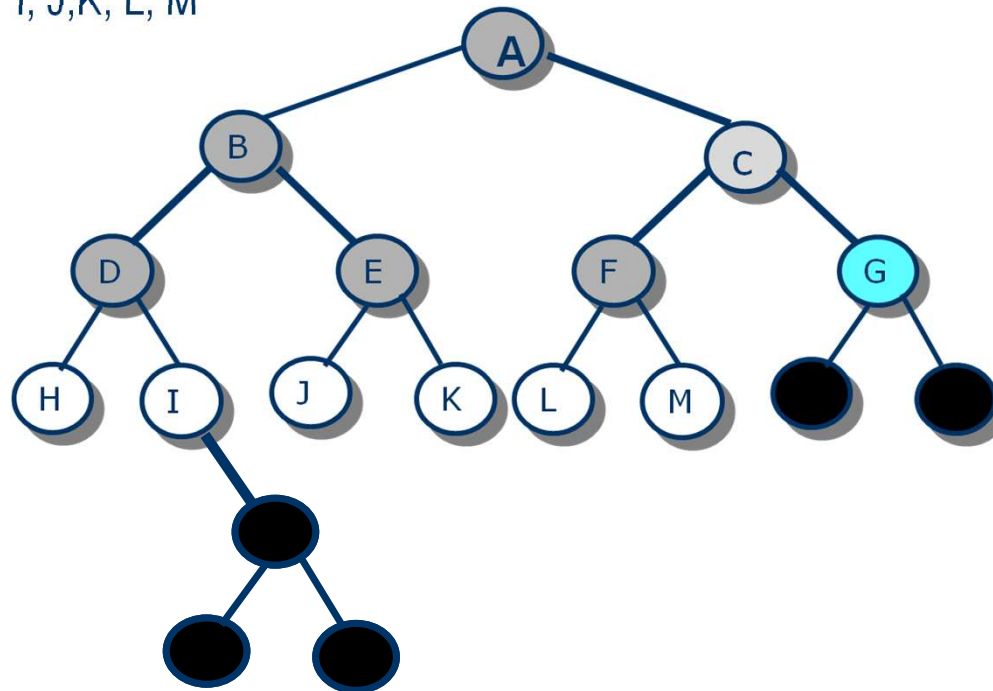


3.1 Búsqueda en Anchura. Ejemplo Genérico

Solución:
A, C, G

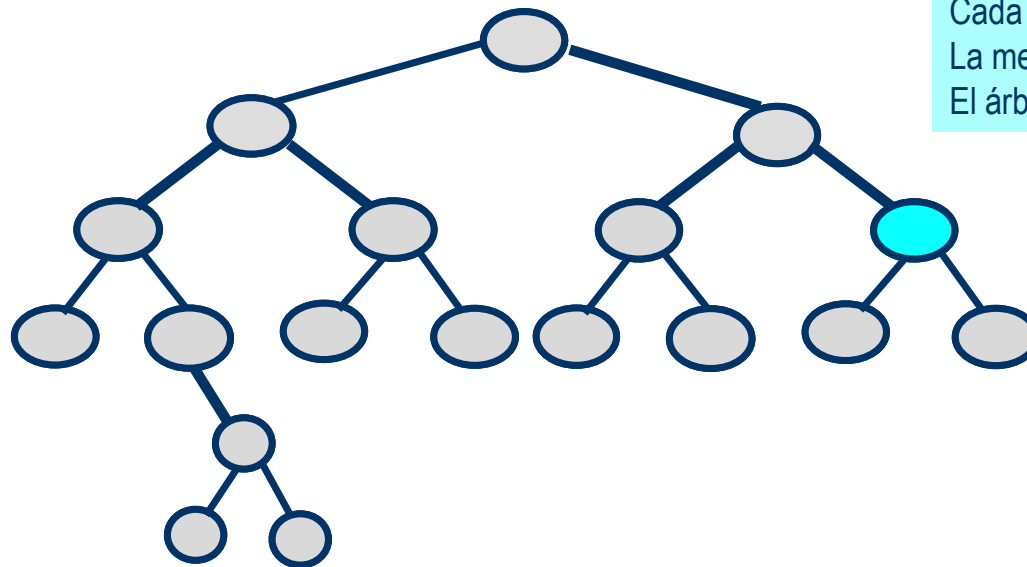
NODOS VISITADOS:
A, B, C, D, E, F, G

NODOS GENERADOS:
A, B, C, D, E, F, G, H,
I, J, K, L, M



3.1 Búsqueda en Anchura. Medidas del Rendimiento

- b : Factor de ramificación, número de nodos generados a partir de cada nodo (máximo número de sucesores de cualquier nodo)
- d : profundidad de la solución óptima
- m : máxima profundidad del árbol



Cada nodo genera 2 nodos: **$b=2$**
La mejor solución se encuentra en el nivel 2: **$d=2$**
El árbol tiene una profundidad máxima de 5: **$m=5$**

3.1 Búsqueda en Anchura. Medidas del Rendimiento

Completa: ¿encuentra una solución?

- Sí, es Completa. Siempre que exista solución, la encontrará en un n° finito de pasos.

Óptima: ¿encuentra la solución óptima?

- Sí, pero cuando el coste del camino a la solución es una función no decreciente de la profundidad del nodo (p.e. cuando todas las acciones tienen el mismo coste).

3.1 Búsqueda en Anchura. Medidas del Rendimiento

Complejidad en tiempo: número de nodos generados

- La raíz genera **b** nodos, cada uno genera **b** nodos, ... :

$$1 + b + b^2 + \dots$$

- Si la solución está a profundidad **d**, en el peor de los casos se han de generar todos excepto el último nodo en el nivel **d** (el objetivo no se expande), por tanto:

$$1 + b + b^2 + \dots + b^d + (b^{d+1} - b) \quad O(b^{d+1})$$

Complejidad en espacio: máximo nº de nodos en memoria

- Si la solución está a profundidad **d**, cada nodo generado debe permanecer en memoria, por tanto en memoria habrá:

$$(1 + b^{d+1} - b) \quad O(b^{d+1})$$

3.1 Búsqueda en Anchura. Análisis

Complejidad en tiempo: $O(b^{d+1})$

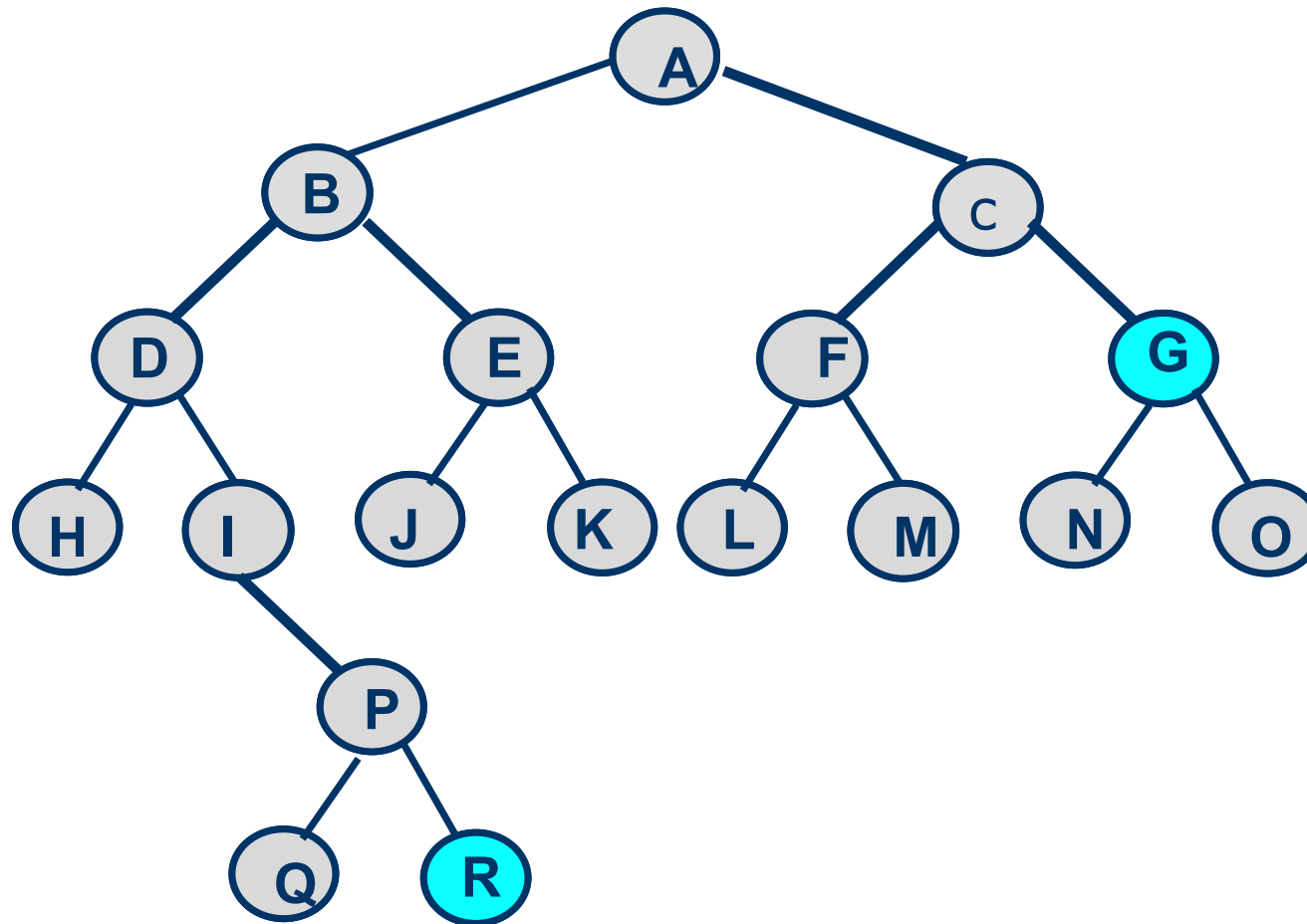
Complejidad en espacio: $O(b^{d+1})$

- El requerimiento de memoria es más importante que el tiempo que tarda en encontrar la solución
- Si existe solución no entrará en bucles infinitos, encontrará la solución
- Ineficaz cuando explora la misma zona del espacio de estados en numerosas ocasiones

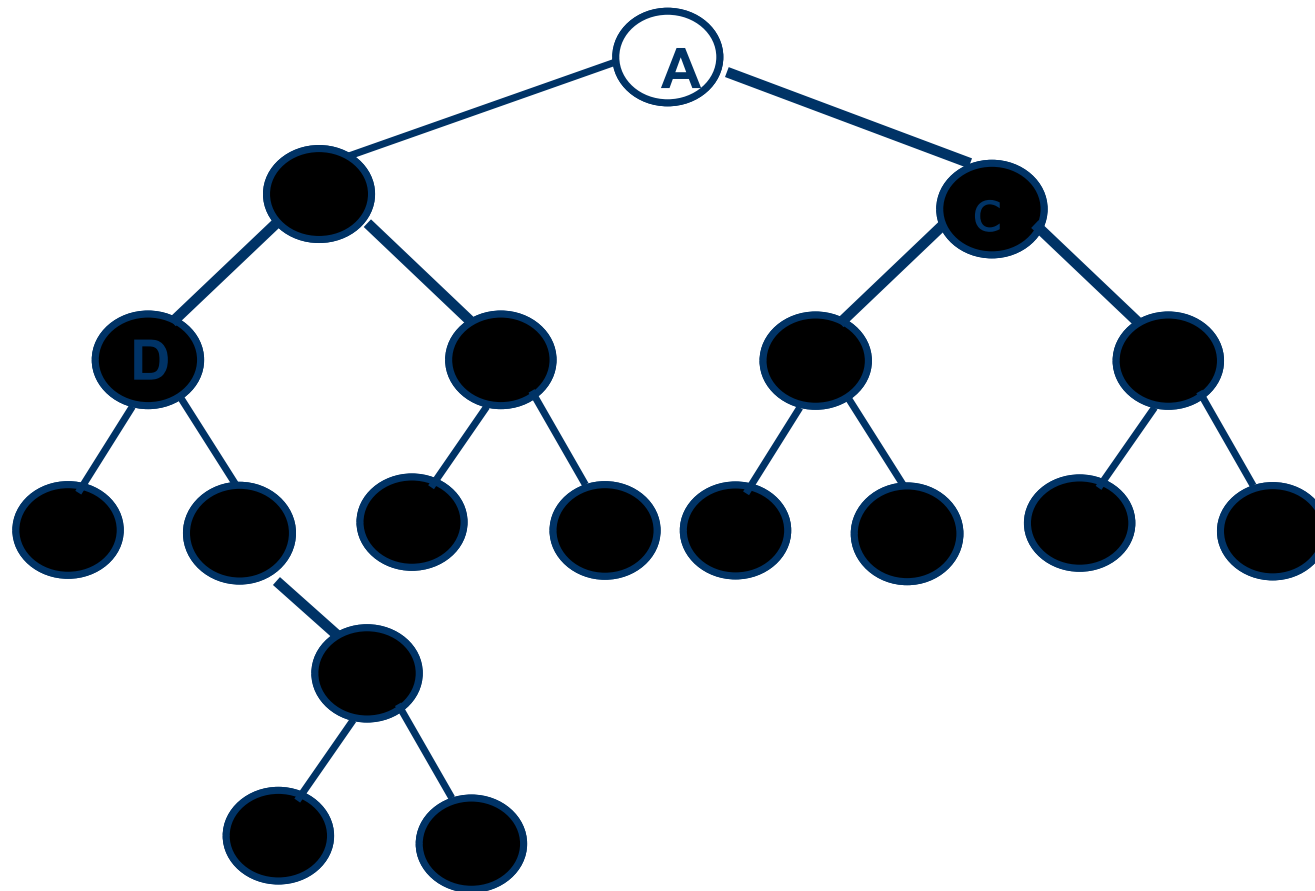
3.2 Búsqueda en Profundidad

- Siempre expande el nodo más profundo
- Sólo cuando la búsqueda encuentra un nodo que no se puede expandir se retrocede para expandir el siguiente nodo más profundo
- Estrategia = seleccionar el nodo más reciente para la expansión (LIFO)

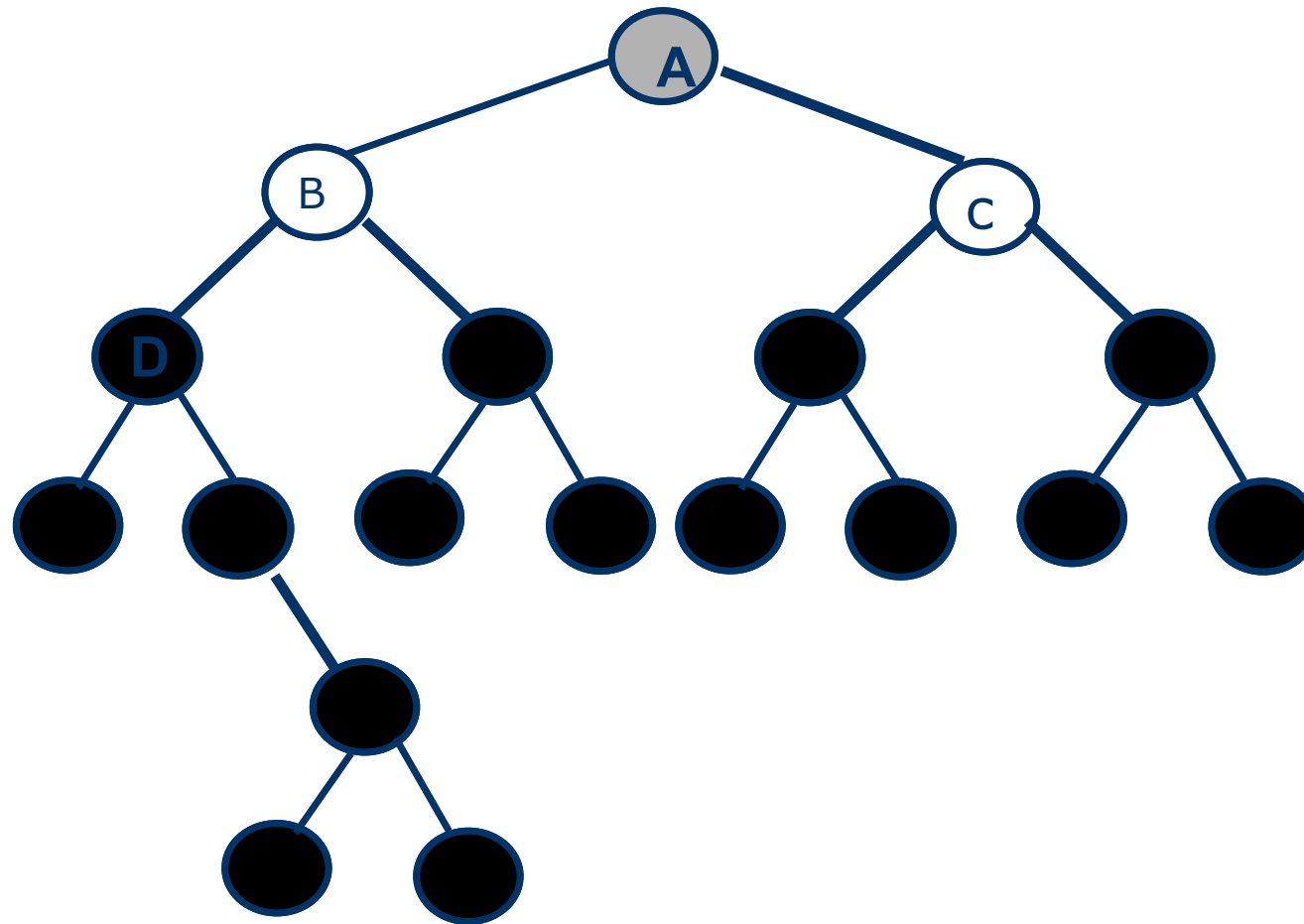
3.2 Búsqueda en Profundidad. Ejemplo Genérico



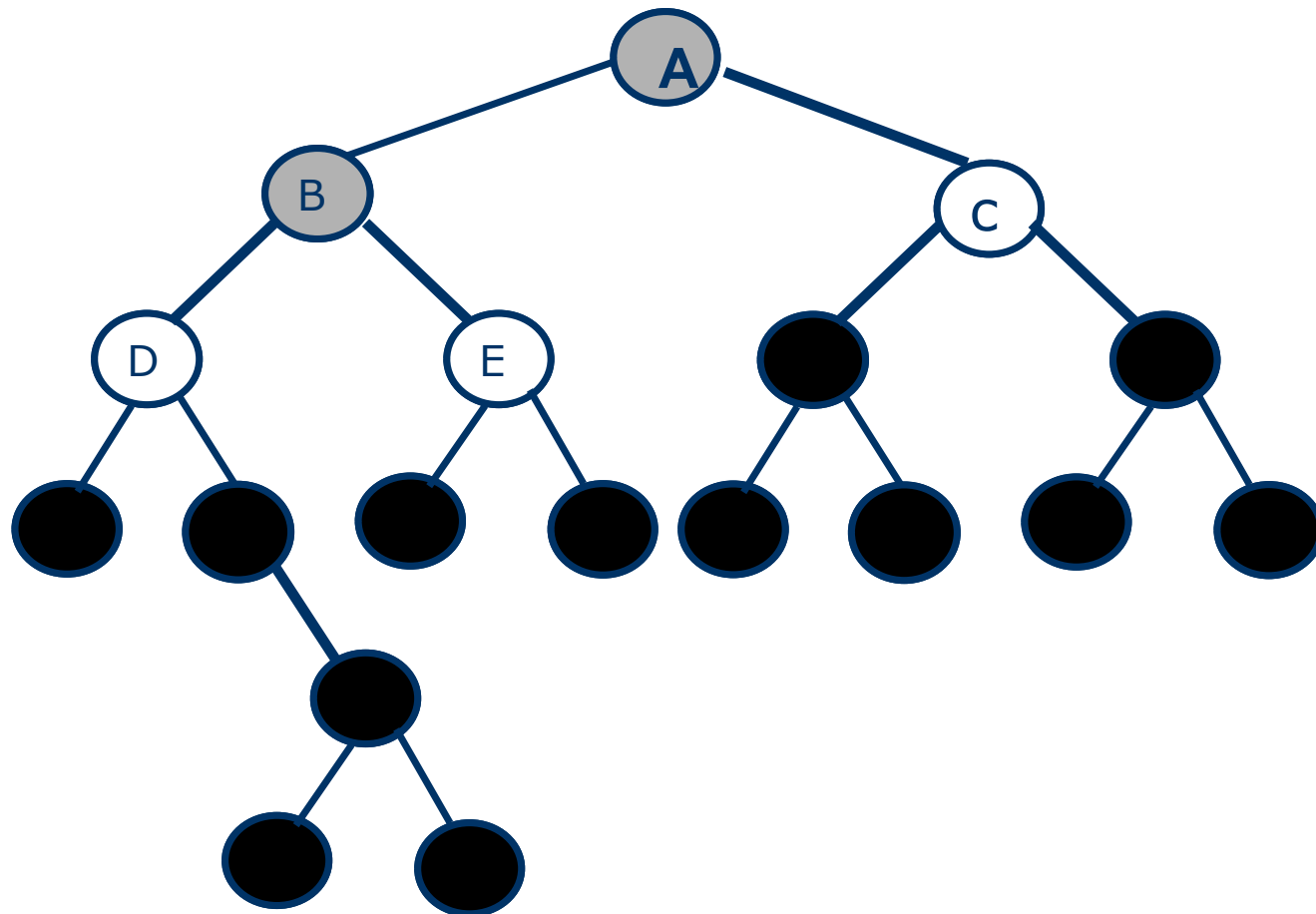
3.2 Búsqueda en Profundidad. Ejemplo Genérico



3.2 Búsqueda en Profundidad. Ejemplo Genérico



3.2 Búsqueda en Profundidad. Ejemplo Genérico

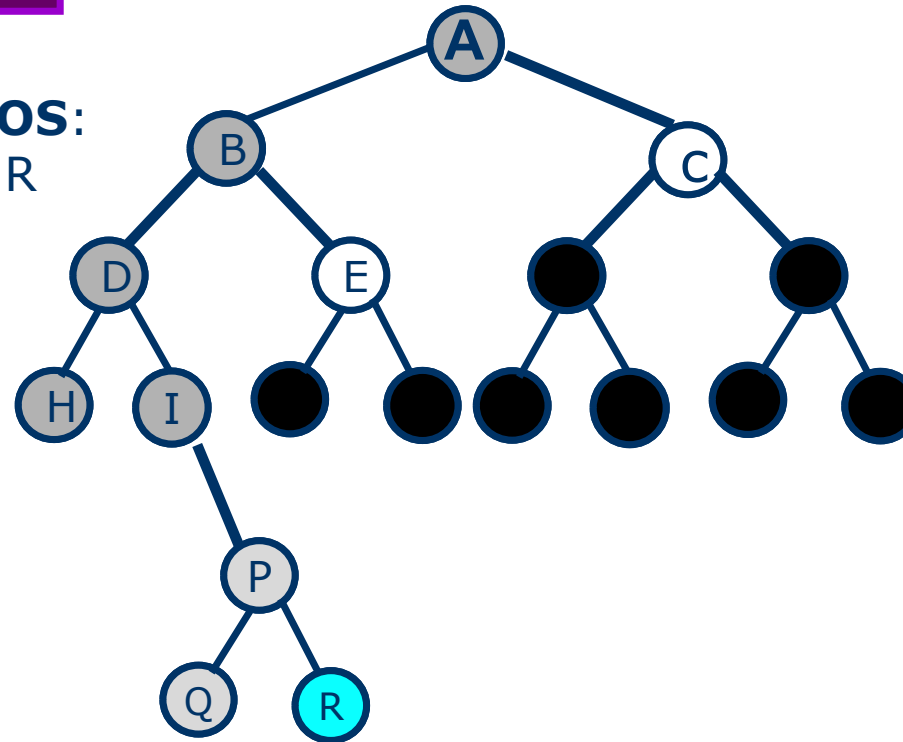


3.2 Búsqueda en Profundidad. Ejemplo Genérico

Solución:
A, B, D, I, P, R

NODOS GENERADOS:
A, B, C, D, E, H, I, P, Q, R

NODOS VISITADOS:
A, B, D, H, I, P, Q, R



3.2 Búsqueda en Profundidad. Medidas del Rendimiento

Completa: ¿encuentra una solución?

- No, no es Completa. Puede perderse en un bucle infinito por una rama por la que no haya solución.

Óptima: ¿encuentra la solución óptima?

- No, no es Óptima. No garantiza que la solución encontrada sea la óptima.

3.2 Búsqueda en Profundidad. Medidas del Rendimiento

Complejidad en tiempo: número de nodos generados

- La raíz genera **b** nodos, cada uno genera **b** nodos, ... :

$$1 + b + b^2 + \dots$$

- Si el árbol tiene una profundidad **m**, en el peor de los casos se ha de llegar hasta la profundidad máxima del árbol, por tanto: $O(b^m)$

Complejidad en espacio: máximo nº de nodos en memoria

- Sólo se necesitan almacenar $(b-1) \cdot m + 1$ nodos por tanto: $O(b \cdot m)$

3.2 Búsqueda en Profundidad. Análisis del Algoritmo

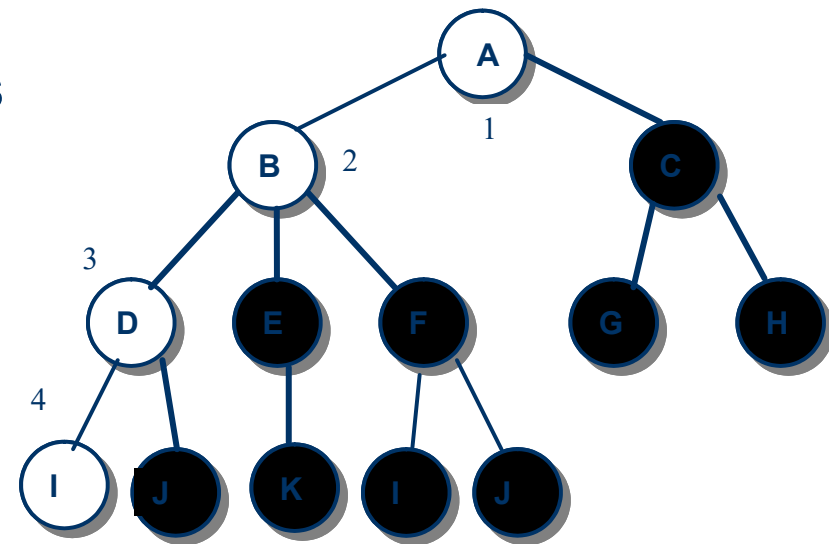
- No es completo (cuando el árbol de búsqueda es infinito)
- Puede caer fácilmente en bucles
- No es óptimo
- Almacena camino desde el nodo raíz a un nodo hoja, junto con los nodos generados pero no expandidos
- Puede encontrar una solución rápidamente

3.3 Estados repetidos

- No volver a un estado del que justo se viene.
- No crear caminos cíclicos.
- No crear un estado que ya ha sido creado antes.
 - Aumenta la efectividad del método.
 - Reduce el tamaño del espacio de estados.
 - Aumenta el coste computacional.
- CERRADOS: nodos expandidos.
- ABIERTOS: nodos generados no expandidos.
- Si el nodo actual se encuentra en Cerrados se elimina en vez de expandirse.

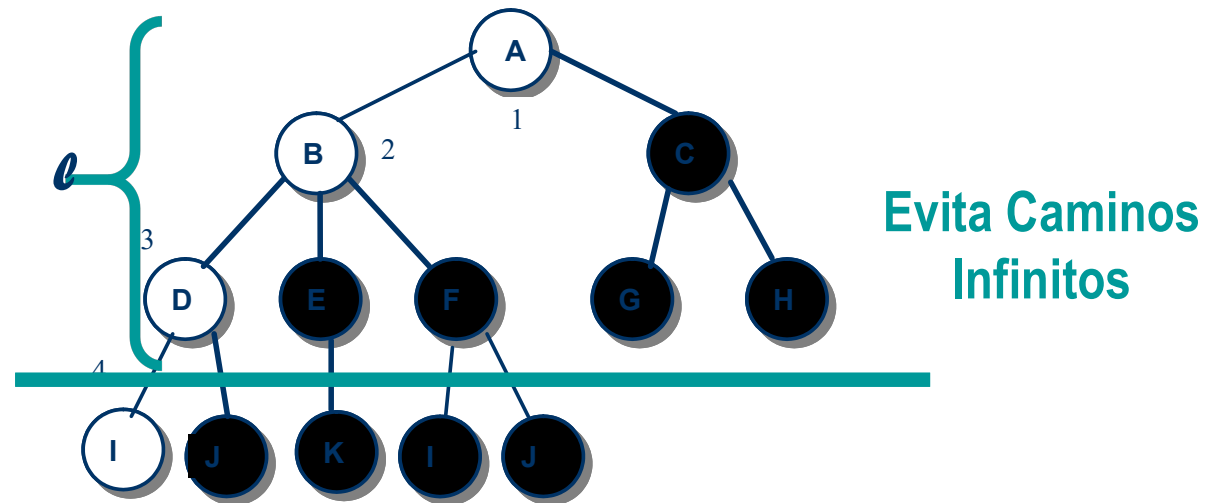
3.4 Variantes. Búsqueda con Retroceso

- Sólo se genera un sucesor a la vez, cada nodo parcialmente expandido recuerda qué sucesor se expande a continuación.
- Complejidad en espacio: $O(m)$
- No apto para grandes descripciones de estados



Menos Memoria

- Poner un límite a la profundidad
 - Incompleta cuando $l < d$
 - No óptima si $l > d$
 - Resuelve el problema de caminos infinitos



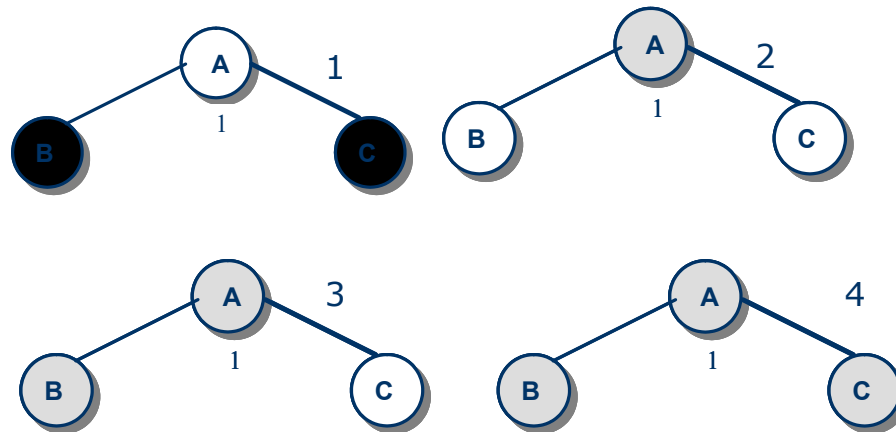
3.4 Variantes. Profundidad Iterativa

- Aumenta gradualmente el límite de profundidad hasta encontrar un objetivo
 - Ventajas de la Búsqueda en Anchura y en Profundidad

Límite = 0



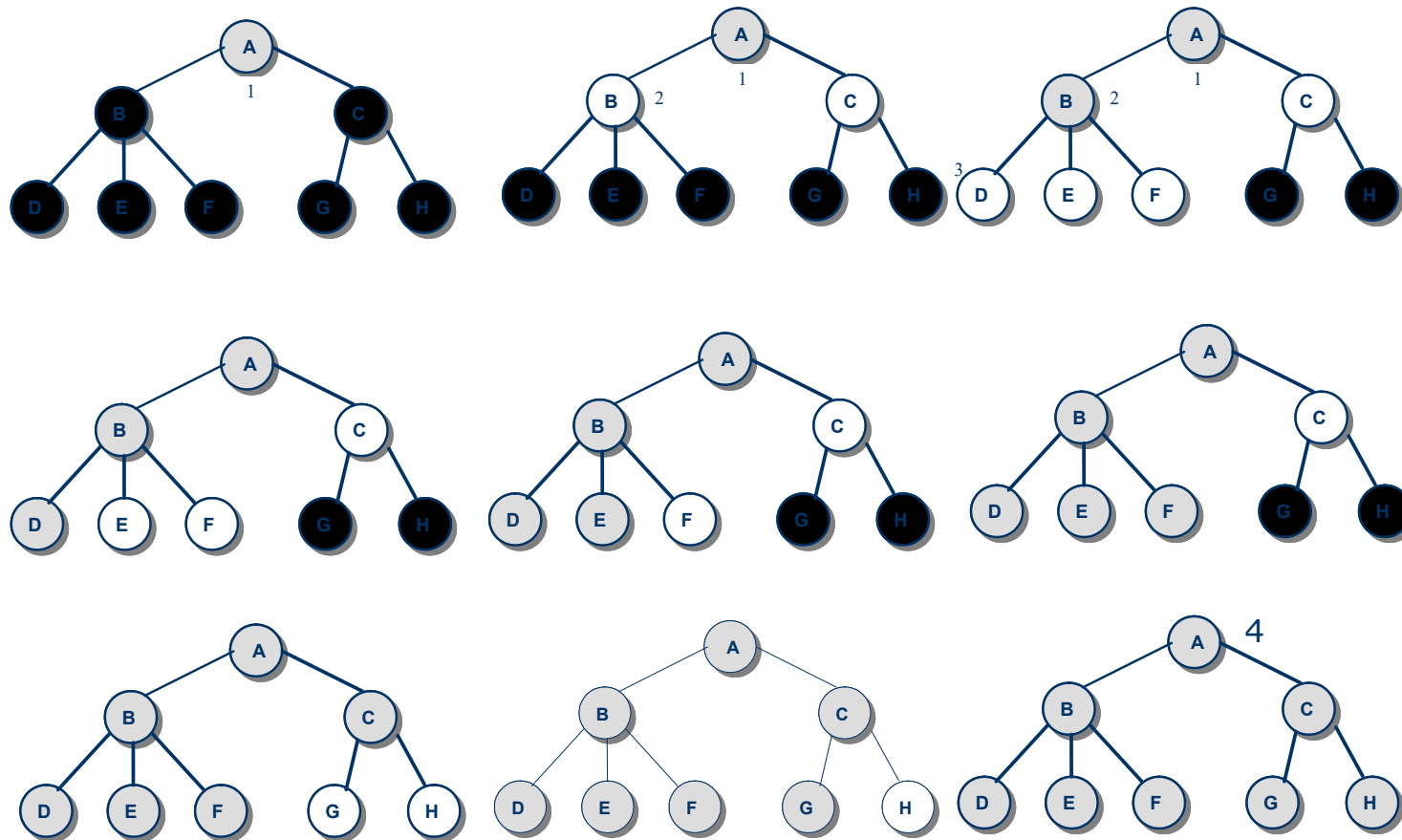
Límite = 1



3.4 Variantes

Profundidad iterativa

Límite = 2



3.4 Variantes.

Búsqueda Bidireccional

- Ejecutar dos búsquedas simultáneas: una hacia delante desde el estado inicial y la otra hacia atrás desde el objetivo, parando cuando las dos búsquedas se encuentren en un estado.
- Función Predecesor
- Búsqueda en Anchura de al menos una de las búsquedas, para garantizar la convergencia del algoritmo.
- Complejidad Temporal: $O(b^{d/2})$
- Complejidad Espacial: $O(b^{d/2})$

3.5 Comparación de Complejidad

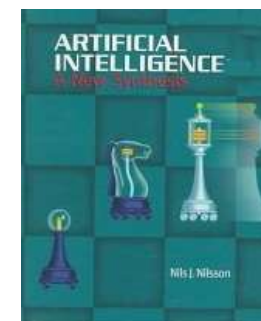
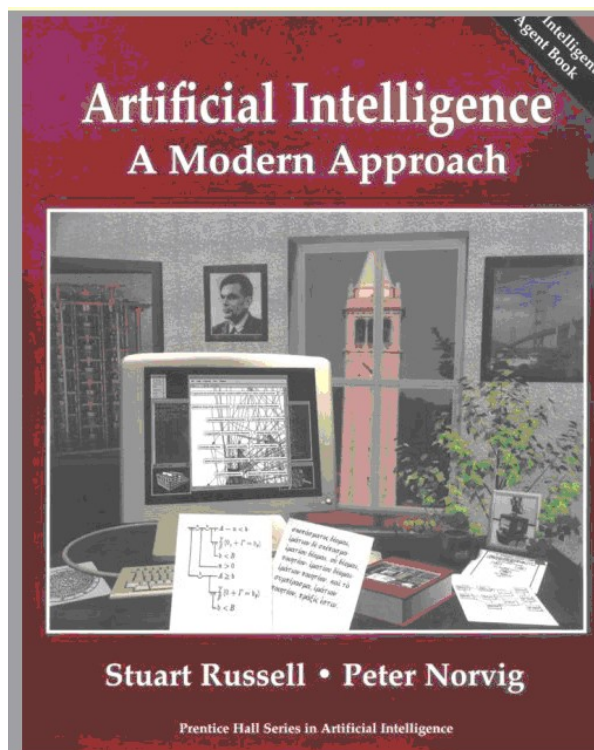
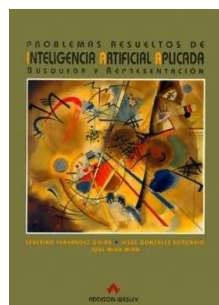
Criterio	1º Anchura	1º Profundidad	Profundidad Limitada	Profundidad Iterativa	Bidireccional
Tiempo	b^d	b^m	b^l	b^d	$b^{d/2}$
Espacio	b^d	bm	bl	bd	$b^{d/2}$
Óptima	Sí*	No	No	Sí	Sí
Completa	Sí	No	Sí, si $l \geq d$	Sí	Sí

*con funciones no decrecientes a la profundidad del nodo, p.e. coste uniforme

b = factor de ramificación
 d = profundidad de la solución
 m = máxima profundidad
 l = límite de profundidad

Referencias Bibliográficas

- **Inteligencia Artificial: Un Enfoque Moderno.** S. Russell y P. Norvig, 2005
- **Problemas Resueltos de IA Aplicada. Búsqueda y Representación.** Fernández et al. (2003)



- **Aspectos básicos de la Inteligencia Artificial.** Mira et al. , 2003

