Inteligencia Artificial

Departamento de Ingeniería Informática

Contenidos del bloque II
Búsqueda en Espacios de Estados

TEMA 2: Agentes basados en objetivos

- Formalización de Problemas de Búsqueda
- Algoritmo genérico de búsqueda

TEMA 3: Búsqueda No Informada

TEMA 4: Búsqueda Informada

TEMA 5: Juegos de 2 adversarios

UCA Universidad de Cádiz

Inteligencia Artificial

Departamento de Ingeniería Informática

TEMA 2. AGENTES BASADOS EN OBJETIVOS

Inteligencia Artificial

2º curso Grado en Ingeniería Informática

Elisa Guerrero Vázquez

Esther L. Silva Ramírez

UCA Universidad de Cádiz

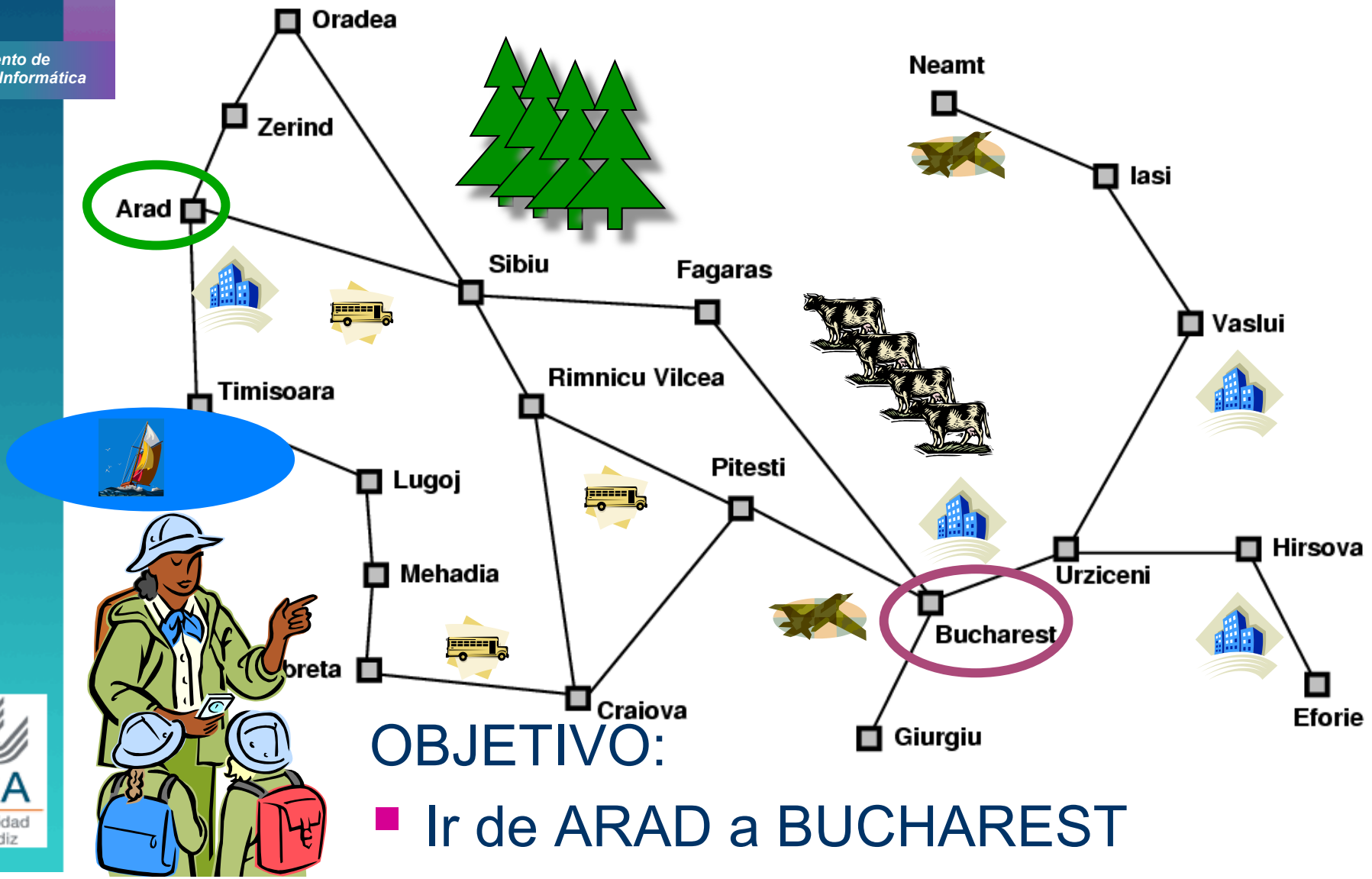
Actuar de forma Racional

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

Introducción

- **Agente**: todo aquello que percibe y actúa en su entorno.
- **Agente ideal**: ante todos los casos de posibles secuencias de percepciones debe emprender todas aquellas acciones que permitan obtener el máximo en su medida de rendimiento. Para ello se basa en las percepciones y en el conocimiento incorporado.
- Pasar de percepciones a acciones, al tiempo que se actualiza un estado interno.
- **Agentes basados en objetivos**: analiza las posibles acciones que se emprenden y elige aquellas que permitan alcanzar el objetivo.
- **Búsqueda**: subcampo de IA que se ocupa de encontrar las secuencias de acciones que permiten alcanzar los objetivos.

Introducción



OBJETIVO:

- Ir de ARAD a BUCHAREST

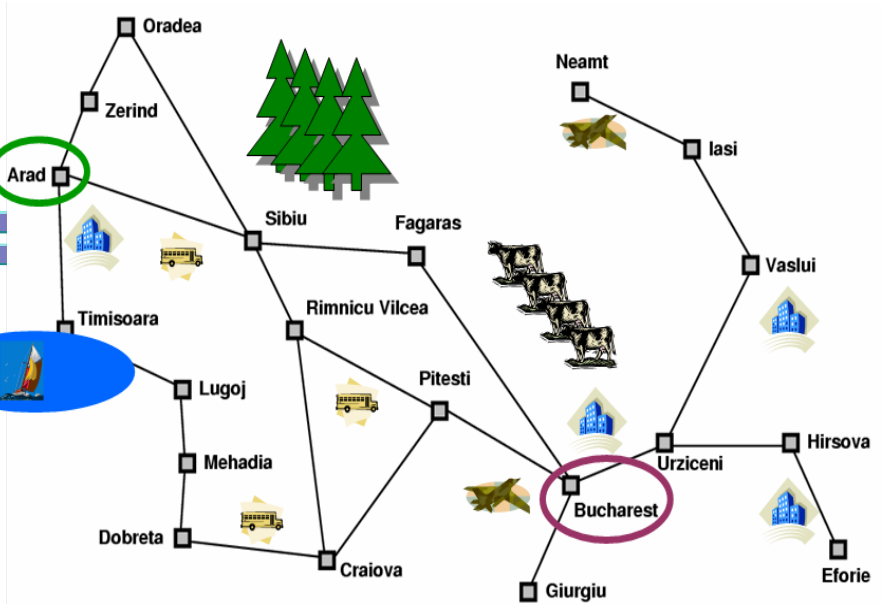
Introducción

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.

Introducción



Objetivo:

- Ir de Arad a Bucharest

Acciones:

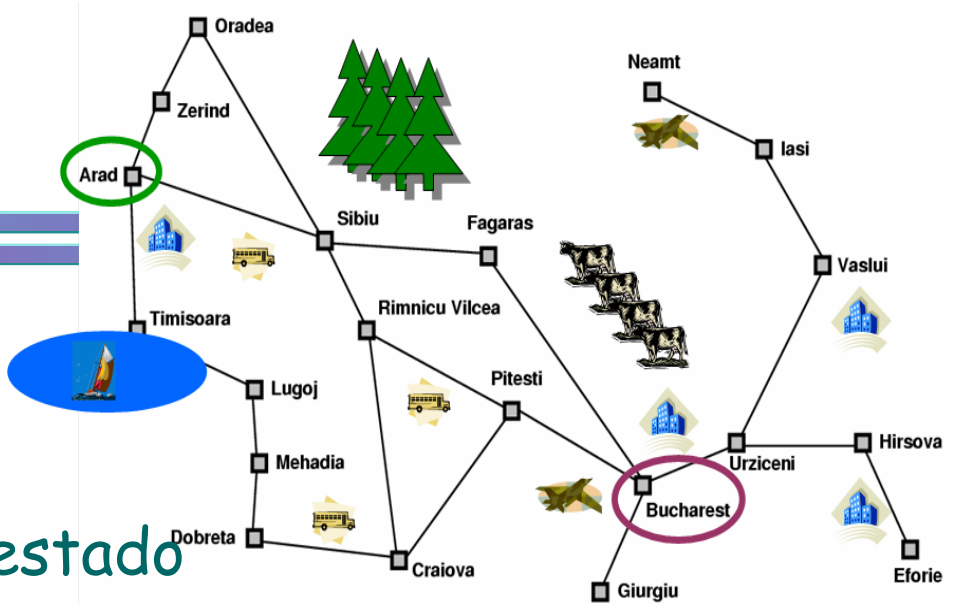
- Viajar por carretera de una ciudad a otra

Estados posibles

- La ciudad donde en cada momento se encuentre el Agente

Los objetivos permiten organizar el comportamiento del agente

Introducción



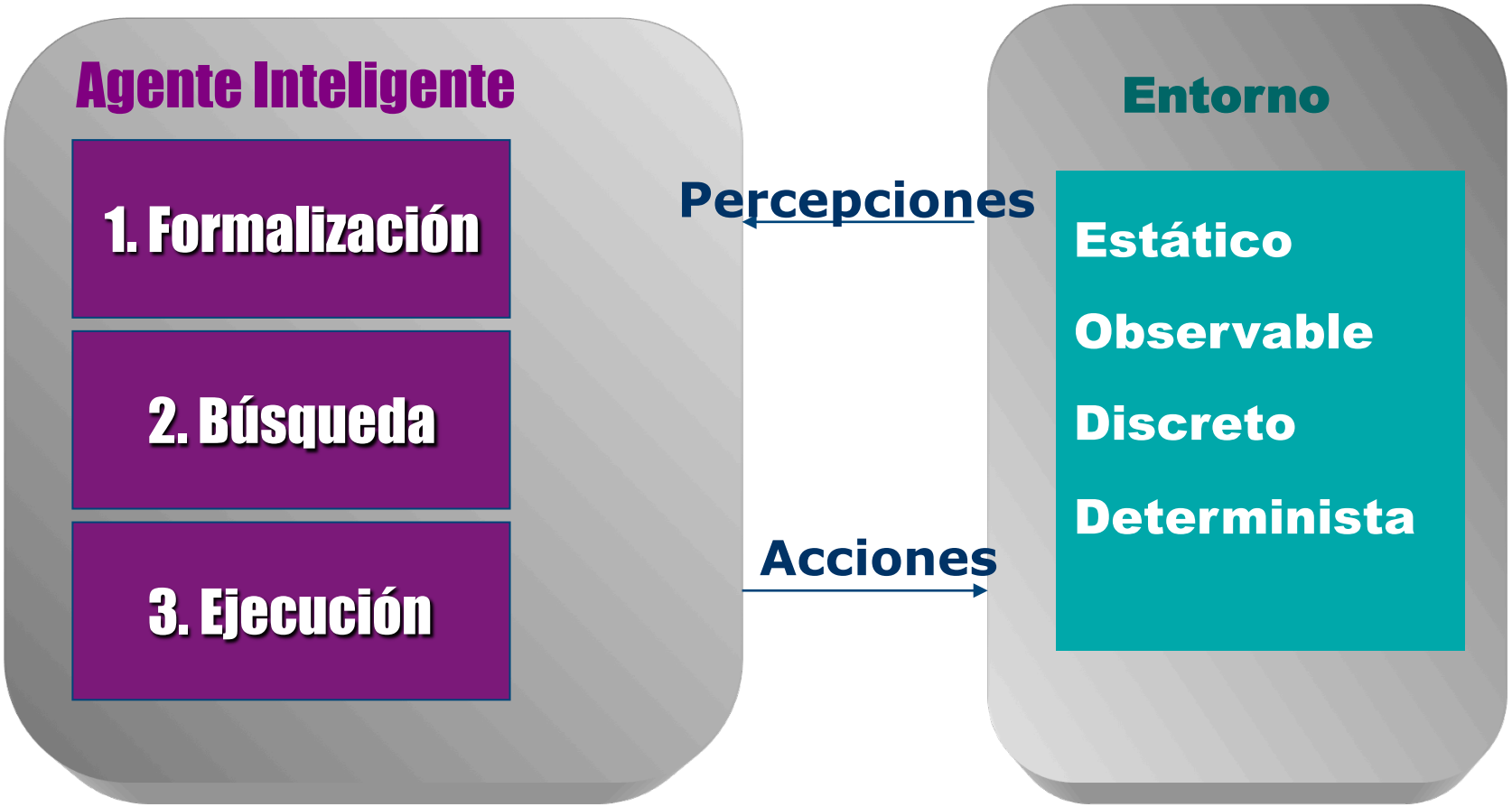
Detalles: Pasos fronterizos, agua, gasolineras, paradas, compañeros de viaje, estado de la carretera, clima, etc.

ABSTRACCIÓN

Eliminar detalles superfluos o no necesarios. Tantos como sea posible mientras se conserve la validez y se asegure que se pueden realizar las acciones

Agentes que resuelven problemas

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



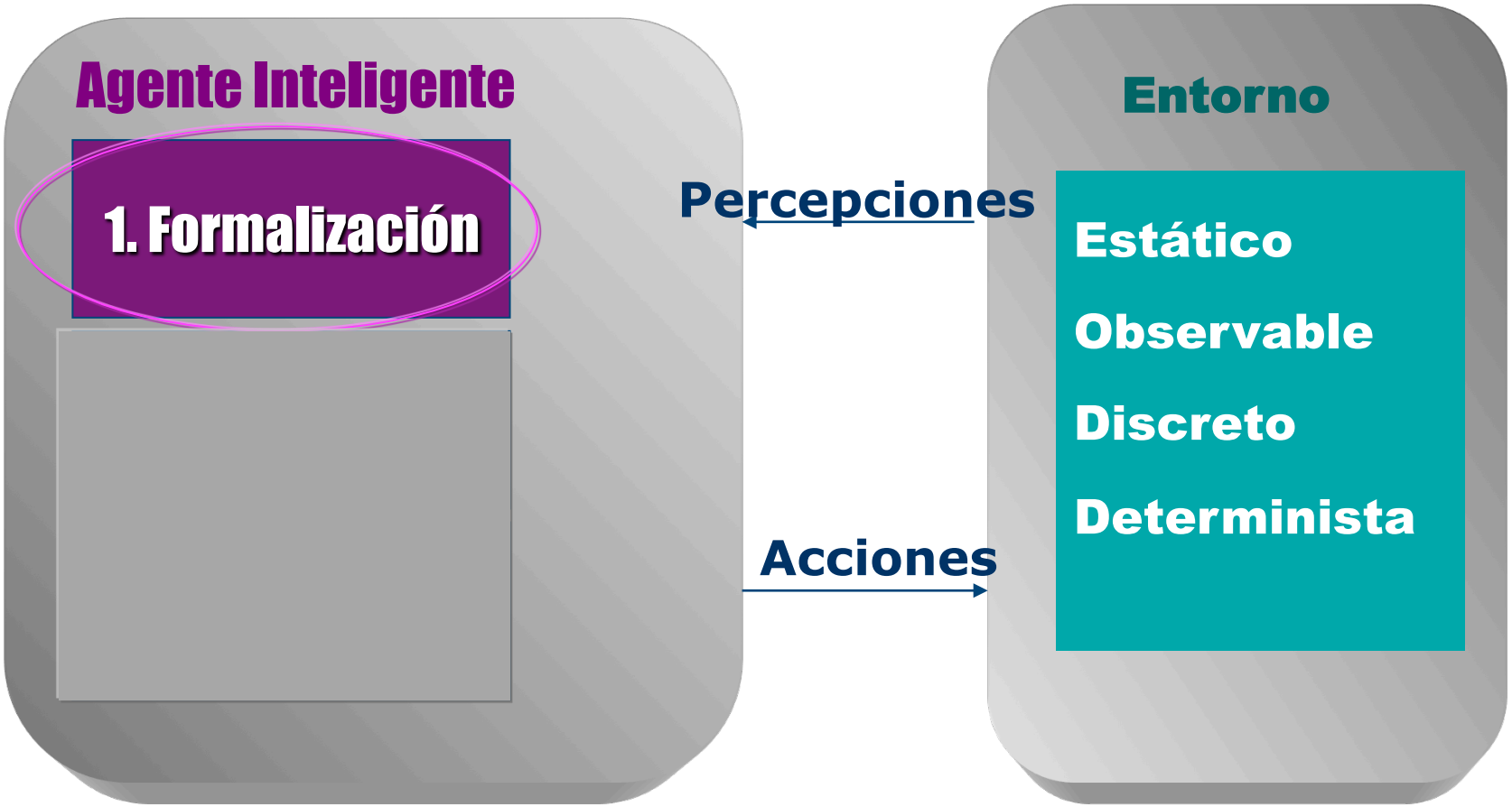
Agentes que resuelven problemas

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

Agentes que resuelven problemas

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



Formalización de un problema de búsqueda

- **Estado Inicial:** situación o configuración inicial desde la que se inicia la resolución del problema
- **Test Objetivo**: verifica si un estado dado es un estado objetivo, una solución al problema
- **Lista de Operadores:** acciones bien definidas
- **Sucesores:** estados a los que se llega como resultado de aplicar los operadores mediante **reglas** (esValido y aplicaOperador), definiéndose el **espacio de estados** del problema a través de un árbol/grafo, los nodos son los estados y la flechas las acciones.
- **Camino:** secuencia de estados conectados por una secuencia de acciones.
- **Coste de la Solución**: medida de rendimiento → Solución óptima

Estado Actual, Sucesores y Operadores

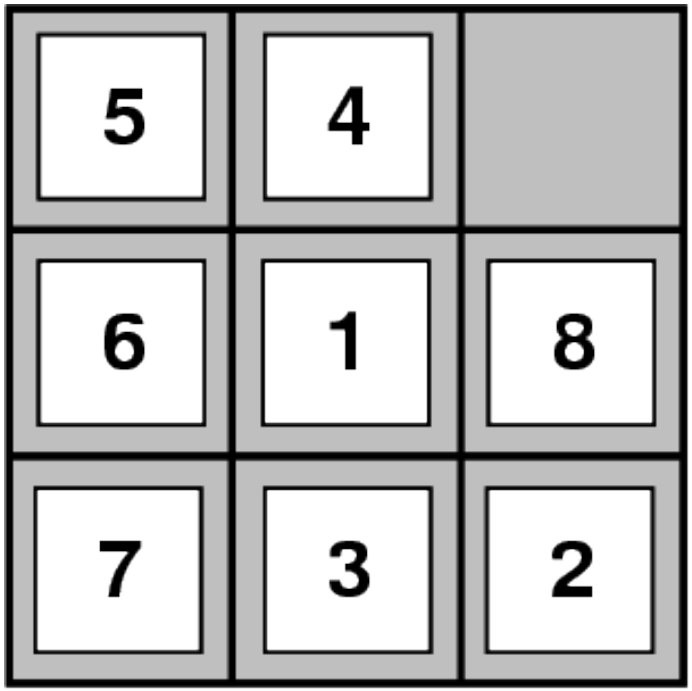
- Los **sucesores** son los nuevos estados válidos que se generan a partir de una determinada situación o estado del problema (el **estado actual**)
- Los operadores son las acciones u operaciones bien definidas mediante reglas que permiten pasar de un estado a otro estado sucesor. Se necesitan dos funciones 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)

Formalización del problema

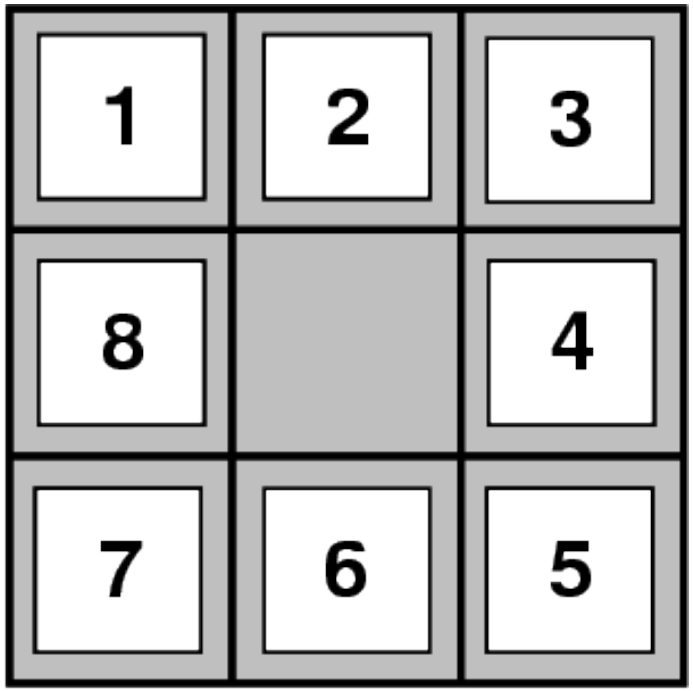
OBJETIVO: Ir a Bucarest

- **Estados:** ciudades donde se puede llegar directamente a partir de la actual
- **Estado Inicial:** Arad
- **Test Objetivo**: ¿Lugar = "Bucarest"?
- **Lista de Operadores:** viajar de una ciudad a otra
- **Sucesores:** nuevas ciudades a la que se llega mediante la regla: "si existe carretera directa entre 2 ciudades conectar ambos lugares", para ello se usan las dos funciones:
  - **esValido**: determinar si existe carretera directa entre la ciudad actual y una nueva ciudad
  - **aplicaOperador:** realizar la acción de viajar de la actual a esta nueva
- **Solución**: ruta por carretera desde Arad a Bucarest
- **Coste de la Solución**: nº de ciudades que visita, o tiempo que tarda en llegar, etc.

Problema del 8-puzzle



Estado Inicial



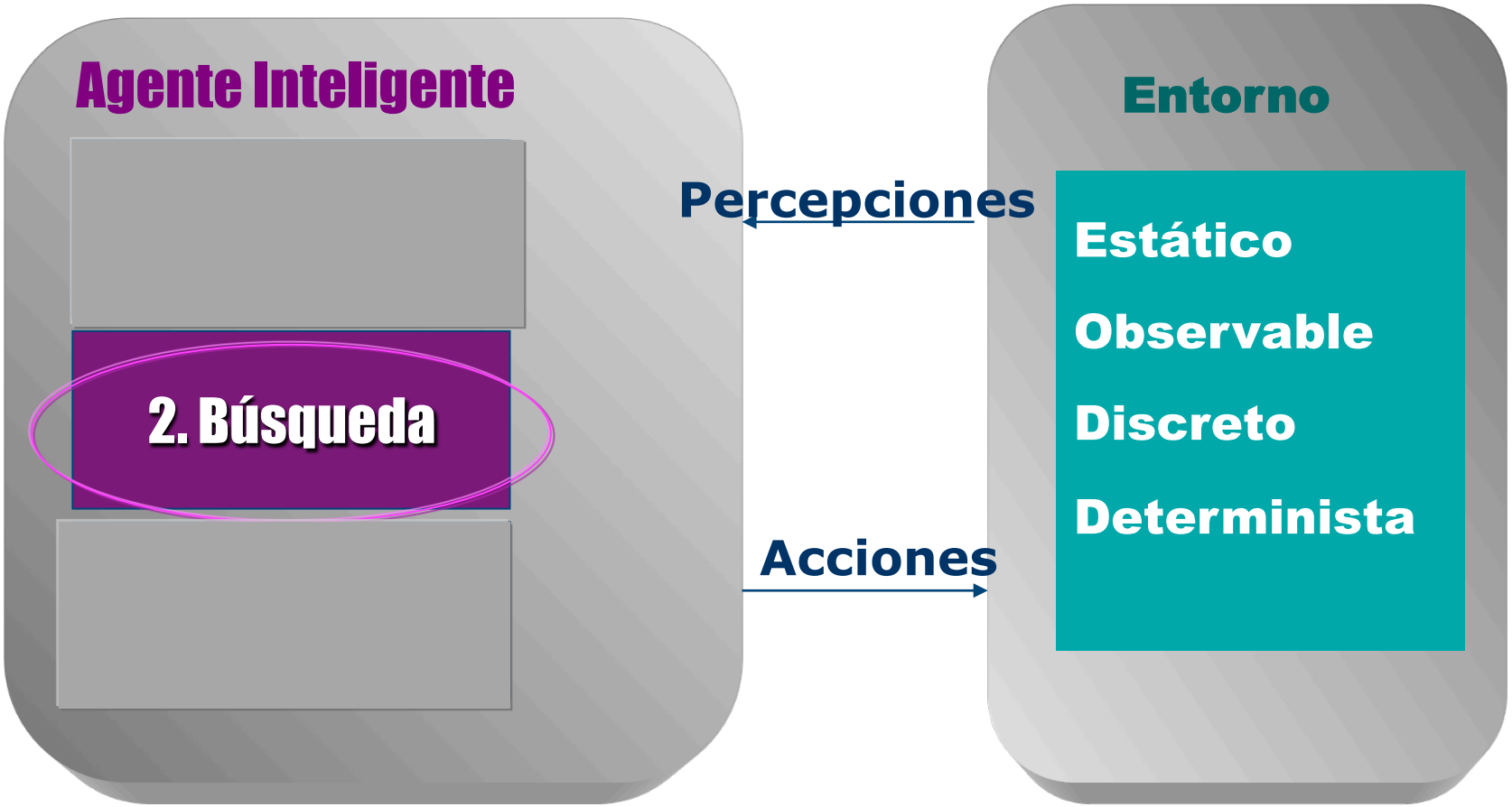
Estado Objetivo

Formalización del 8-puzzle

- **Estados**: Tableros o puzles generados al intercambiar la ficha vacía por otra ficha
- **Estado inicial**: cualquier configuración de fichas
- **Test objetivo**: comprobar si la disposición de las fichas de un estado coincide con el estado final
- **Lista de Operadores:** Mover hueco a la derecha, a la izquierda, arriba o abajo
- **Sucesores:** nuevos estados a la que se llega mediante la regla: "mover el hueco a una posición adyacente teniendo en cuenta los límites del tablero", para ello 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: i**ntercambiar la ficha vacía con otra ficha que se encuentre en una posición adyacente válida, aplicando uno de los 4 operadores.
- **Solución**: Movimientos para llegar desde estado inicial al estado objetivo
- **Coste del camino**: coste de cada paso

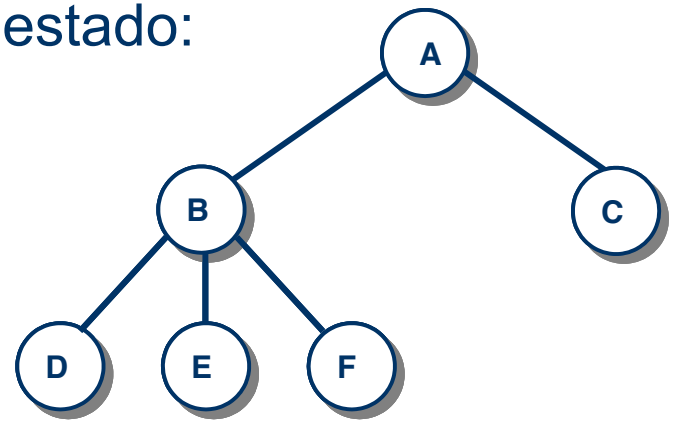
Agentes que resuelven problemas

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



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?
- Expandir: generación de todos los sucesores del nodo actual cuando no es un nodo objetivo
- Lista de Nodos ABIERTOS
- Lista de Nodos CERRADOS

Estrategias de búsqueda

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

Algoritmo General de Búsqueda

1. El nodo inicial se guarda en la lista de nodos Abiertos
2. Mientras no se haya llegado al objetivo y Abiertos no esté Vacía hacer:

   2.1 Seleccionar primer nodo de Abiertos, que llamaremos Nodo Actual

   2.2 Si este Actual no es el Objetivo

      2.2.1 Expandir: genera la lista de sucesores **válidos** que pueden obtenerse a partir del nodo Actual

      2.2.2 La lista de Sucesores se añade a la lista de Abiertos de acuerdo a algún criterio
3. Finaliza dando el camino a la solución o con mensaje de que no se ha encontrado

Algoritmo General de Búsqueda

```
Solucion: función Búsqueda (tNodo: Inicial, entero: estrategia)
inicio
  tNodo Actual
  tLista: Abiertos ← {Inicial}  // El nodo inicial se guarda en Abiertos
  logico Objetivo: Falso
  mientras (No Vacia(Abiertos)) Y (No Objetivo)
          Actual ← Primero(Abiertos) // selecciona primer nodo de Abiertos
          si EsObjetivo(Actual) entonces
                  Objetivo ← Verdadero
          si_no
                  Sucesores ← Expandir(Actual)
                  Abiertos ← {Abiertos+Sucesores} //de acuerdo a estrategia
          fin_si
  fin_mientras
  si Objetivo entonces
       devolver  Camino a la Solución
  si_no devolver Fallo
fin_función
```

Función Expandir

- **La función Expandir busca los posibles sucesores que puede tener un nodo y los almacena en la lista Sucesores**

```
tLista: función Expandir(tNodo: actual)
inicio
        tNodo: nuevo
        tLista: Sucesores← { }
        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
fin_función
```

Medidas del Rendimiento

Completa: la estrategia siempre que exista, encontrará una solución

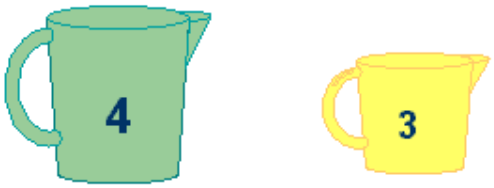
Óptima: la estrategia siempre que exista solución, encontrará primero la mejor solución

Complejidad en tiempo: número de nodos generados durante la búsqueda

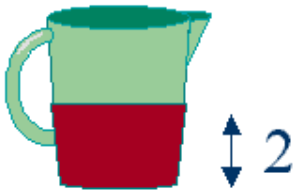
Complejidad en espacio: máximo número de nodos en memoria

El Problema de las Jarras de Vino

- **Entrada:** dos garrafas vacías de vino de 4 y 3 litros



- **Salida**: la garrafa de 4 litros contiene 2



Operaciones permitidas:

- **Llenar** las garrafas del depósito
- **Vaciar** las garrafas en el depósito
- **Pasar** contenido de una garrafa a otra hasta que una se vacíe o se llene la otra

- **Medios**: depósito con vino suficiente

Formalización de un problema de búsqueda

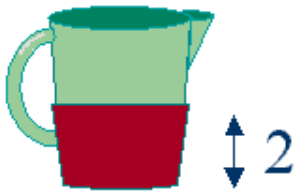
- **Estados**: contenido de cada jarra (variables de tipo entero)
- **Estado Inicial:** Jarras Vacías
- **Test Objetivo**: La jarra de 4 litros tiene exactamente 2 litros (el contenido de la jarra de 3 litros es indiferente)
- **Lista de Operadores:** operaciones de llenar, pasar y vaciar (se enumeraran cada una de ellas)
- **Sucesores:** contenidos de las jarras después de realizar las acciones correspondientes,
  - **esValido**: si se puede pasar contenido o vaciar o llenar del depósito, la acción será válida
  - **aplicaOperador:** pasar el contenido o vaciar o llenar del depósito
- **Solución:** secuencia de acciones que proporcionan los distintos estados que llevan al objetivo.
- **Coste de la Solución**: medida de rendimiento → Solución óptima

Formalización (Jarras de Vino)

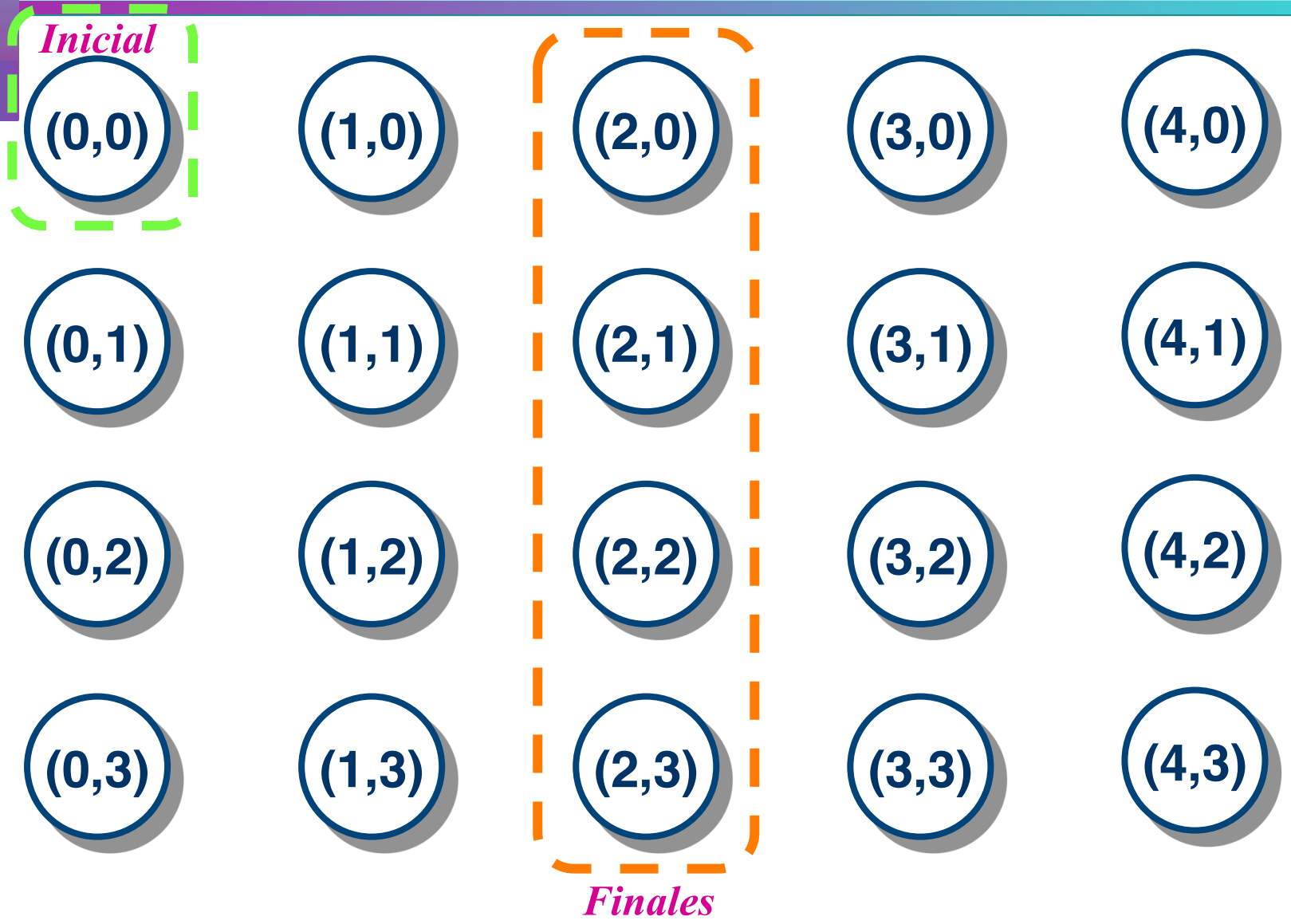
- Espacio de Estados = (**x**, **y**)

  tal que **x** ∈ {0,1,2,3,4}, **y** ∈ {0,1,2,3}

- Estado Inicial : (0, 0)
- Estado Final : (2, n)



Espacio de Estados (jarras)



Operaciones (jarras)

- Llenar la menor (x,y) -> (x,3)
- Llenar la mayor (x,y) -> (4,y)
- Vaciar la menor (x,y) -> (x,0)
- Vaciar la mayor (x,y) -> (0,y)

- Pasar de la menor a la mayor hasta que se llene o se vacíe la menor
- Pasar de la mayor a la menor hasta que se llene o se vacíe la mayor

Restricciones (jarras)

- No tiene sentido llenar una jarra que ya está llena

  Si y<3, (x,y) -> (x,3)

  Si x<4, (x,y) -> (4,y)

- No tiene sentido vaciar una jarra que ya está vacía

  Si y>0, (x,y) -> (x,0)

  Si x>0, (x,y) -> (0,y)

Conjunto de Reglas (jarras)

- 1. Llenar J4 (x,y), x<4 -> (4,y)
- 2. Llenar J3 (x,y), y<3 -> (x,3)
- 3. Vaciar J4 (x,y), x>0 -> (0,y)
- 4. Vaciar J3 (x,y), y>0 -> (x,0)
- 5. Llenar J4 con J3
- 6. Llenar J3 con J4
- 7. Vaciar J3 en J4
- 8. Vaciar J4 en J3

Conjunto de Reglas (jarras)

- 1. Llenar J4 (x,y), $x<4$ -> (4,y)
- 2. Llenar J3 (x,y), $y<3$ -> (x,3)
- 3. Vaciar J4 (x,y), $x>0$ -> (0,y)
- 4. Vaciar J3 (x,y), $y>0$ -> (x,0)
- 5. Llenar J4 con J3

  (x,y), $x<4$, $x+y\geq 4$, $y>0$ -> $(4,x+y-4)$
- 6. Llenar J3 con J4

  (x,y), $y<3$, $x+y\geq 3$, $x>0$ -> $(x+y-3,3)$
- 7. Vaciar J3 en J4

  (x,y), $x+y\leq 4$, $y>0$ -> $(x+y,0)$
- 8. Vaciar J4 en J3

  (x,y), $x+y\leq 3$, $x>0$ -> $(0,x+y)$

Pseudocódigo para la formalización

```
const
    // cada operador llevará asociado un número constante
tipo
    // se define el contenido del tipo tEstado

// funciones útiles: crear estado inicial, final, iguales, etc.

lógico función testObjetivo(tEstado: s)
lógico función esValido(entero: op, tEstado: s)
entero función aplicaOperador(entero: op, tEstado: s)
```

Estas 3 funciones tendrán siempre el mismo prototipo, pero su implementación concreta depende del problema particular planteado. De esta manera se podrá reutilizar el código de búsqueda con independencia del problema

Operadores

- Cada nombre de operador tiene asignado un número constante:

```
const
    LlenarJ4 = 1
    LlenarJ3 = 2
    VaciarJ4 = 3
    VaciarJ3 = 4
    LlenarJ4conJ3 = 5
    LlenarJ3conJ4 = 6
    Vaciar J4enJ3 = 7
    VaciarJ3enJ4 = 8

    NUM_OPERADORES=8
```

Tipo de datos del estado

- Se define un tipo de datos específico para representar el estado

```
tipo
  registro: tEstado
        entero: x, y
  fin_registro
```

Mantendremos siempre una estructura tEstado, pero su contenido cambiará de acuerdo al problema concreto
Así podremos reutilizar el código de búsqueda con independencia del problema

Función testObjetivo

```
lógico: función testObjetivo(E tEstado: estado)
inicio
   devolver estado.x=2
fin_función
```

Función esValido

```
lógico: función esValido(E entero: op, E tEstado: estado)
var
   lógico: valido
Inicio
    según_sea (op) hacer
          llenarJ4: valido ← estado.x<4
          llenarJ3: valido ← estado.y<3
          vaciarJ4: valido ← estado.x>0
          vaciarJ3: valido ← estado.y>0
```

Función esValido

llenarJ4conJ3: valido ← (estado.x+estado.y)≥4 ∧ (estado.x<4) ∧ (estado.y>0)

llenarJ3conJ4: valido ← (estado.x+estado.y≥3) ∧ (estado.y<3) ∧ (estado.x>0)

vaciarJ4enJ3: valido ← (estado.x+estado.y≤3) ∧ (estado.x>0) ∧ (estado.y<3)

vaciar3en4: valido ← (estado.x+estado.y≤4) ∧ (estado.y>0) ∧ (estado.x<4)

en_otro_caso: valido ← falso

fin_según

devolver valido

fin_función

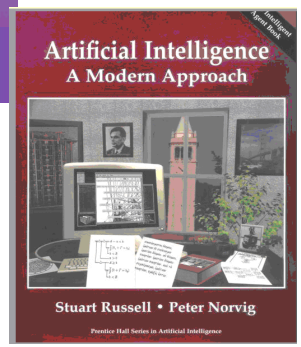
Función aplicaOperador

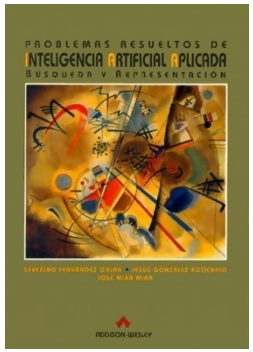
```
tEstado: función aplicaOperador(E entero: op, E tEstado: estado)
var
    tEstado: nuevo
inicio
    nuevo ← estado
     según_sea (op) hacer
          llenarJ4: nuevo.x=4
          llenarJ3: nuevo.y=3
          vaciarJ4: nuevo.x=0
          vaciarJ3: nuevo.y=0
          llenarJ4conJ3: nuevo.x=4                    nuevo.y=4-estado.x
          llenarJ3conJ4: nuevo.x=3-estado.y           nuevo.y=3
          vaciarJ4enJ3:  nuevo.x=0          nuevo.y=estado.y+estado.x
          vaciarJ3enJ4:  nuevo.x=estado.x+estado.y    nuevo.y=0
     fin_según
     devolver nuevo
fin_función
```

Coste de la Solución

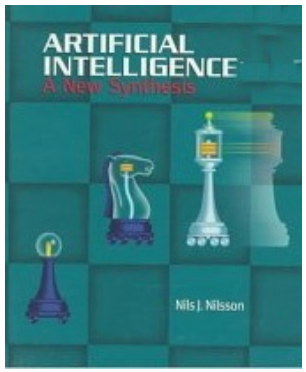
- Alternativas
  - Sin coste: cualquier solución es válida
  - Coste =1: hay que buscar soluciones con el menor nº de acciones (llenar, vaciar, ...)

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