



TEMA 2. AGENTES BASADOS EN OBJETIVOS

Inteligencia Artificial
3º curso Grado en Ingeniería Informática
Elisa Guerrero Vázquez

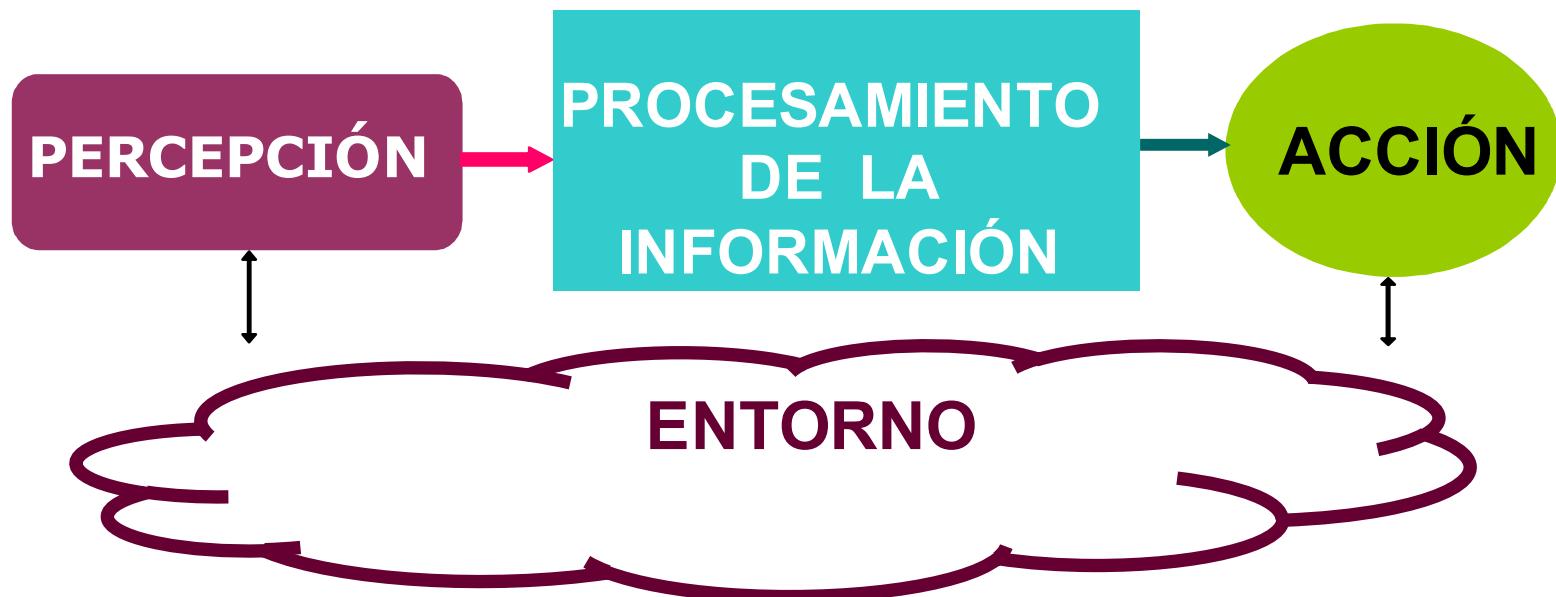


Objetivos

- Al finalizar el tema 2 el alumno ha de ser capaz de:
 - Conocer el funcionamiento básico de una estrategia de búsqueda en espacios de estados.
 - Identificar los tipos de problemas que pueden ser resueltos mediante estrategias de búsqueda en espacios de estados.
 - Realizar la formalización de problemas de búsqueda utilizando la información específica del dominio del problema.
 - Implementar en C dichas formalizaciones.

Actuar de forma Racional

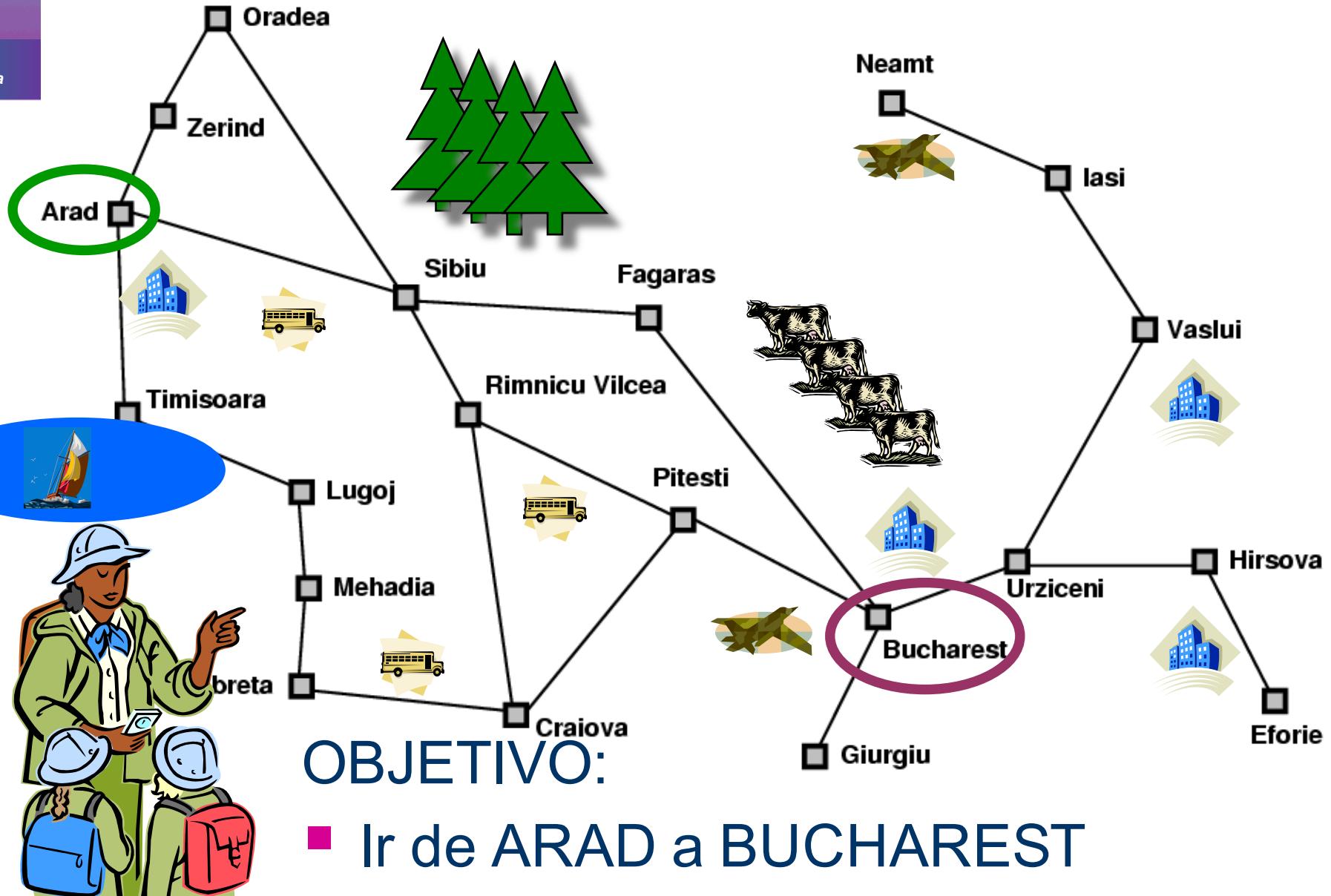
- Un **agente inteligente** es aquél que emprende la mejor acción posible ante una situación dada.
(Russell & Norvig, 2004)



Introducción

- **Agente:** todo aquello que percibe y actúa en su entorno.
- **Agentes basados en objetivos:** analiza las posibles acciones que se pueden emprender 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



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:

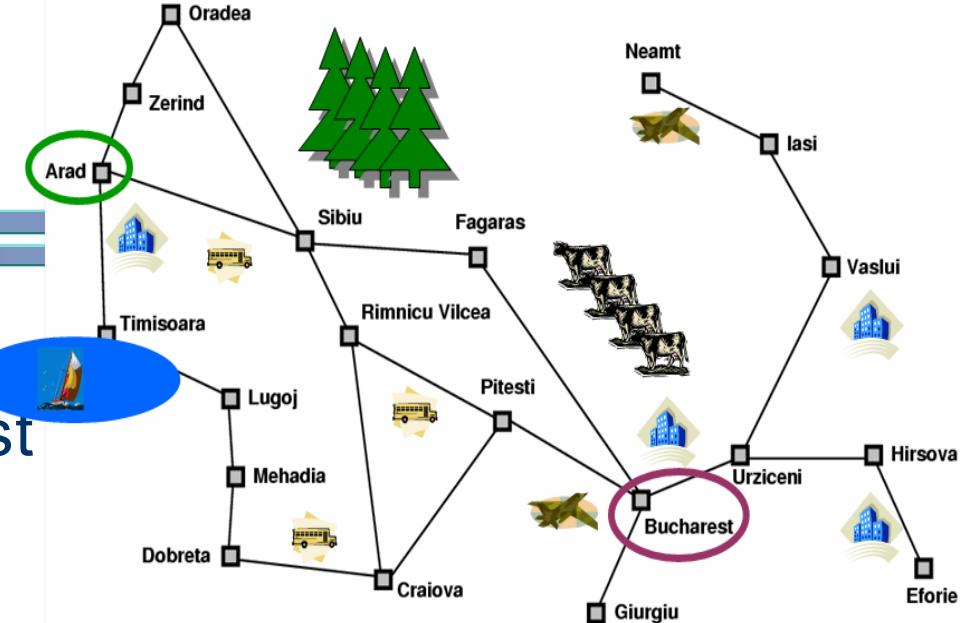
- Ir de Arad a Bucharest

Acciones:

- Viajar por carretera de una ciudad a otra

Estados posibles

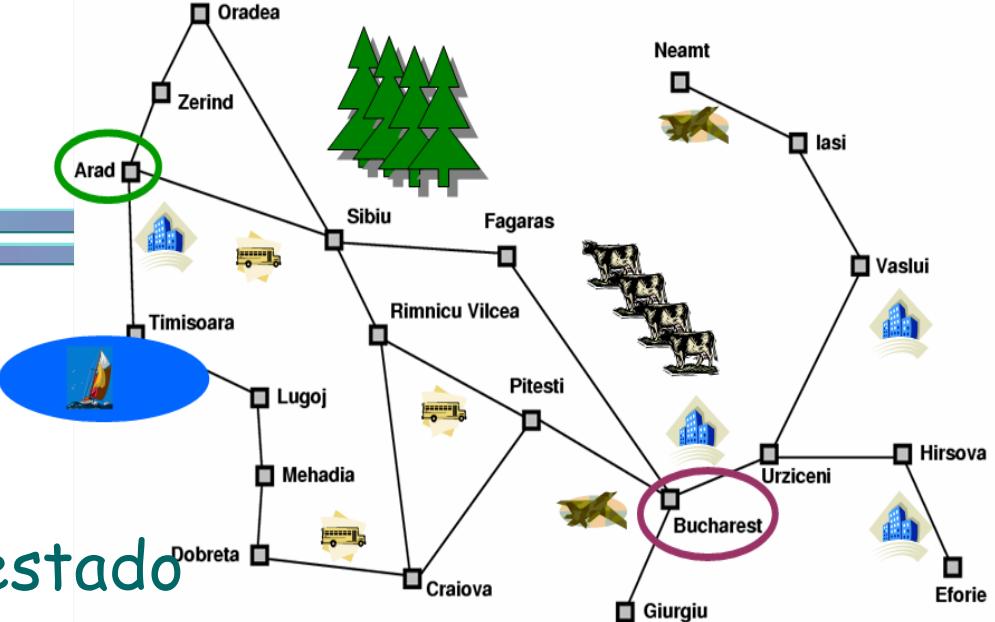
- La ciudad donde en cada momento se encuentre el Agente



**Los objetivos permiten organizar
el comportamiento del agente**

Introducción

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

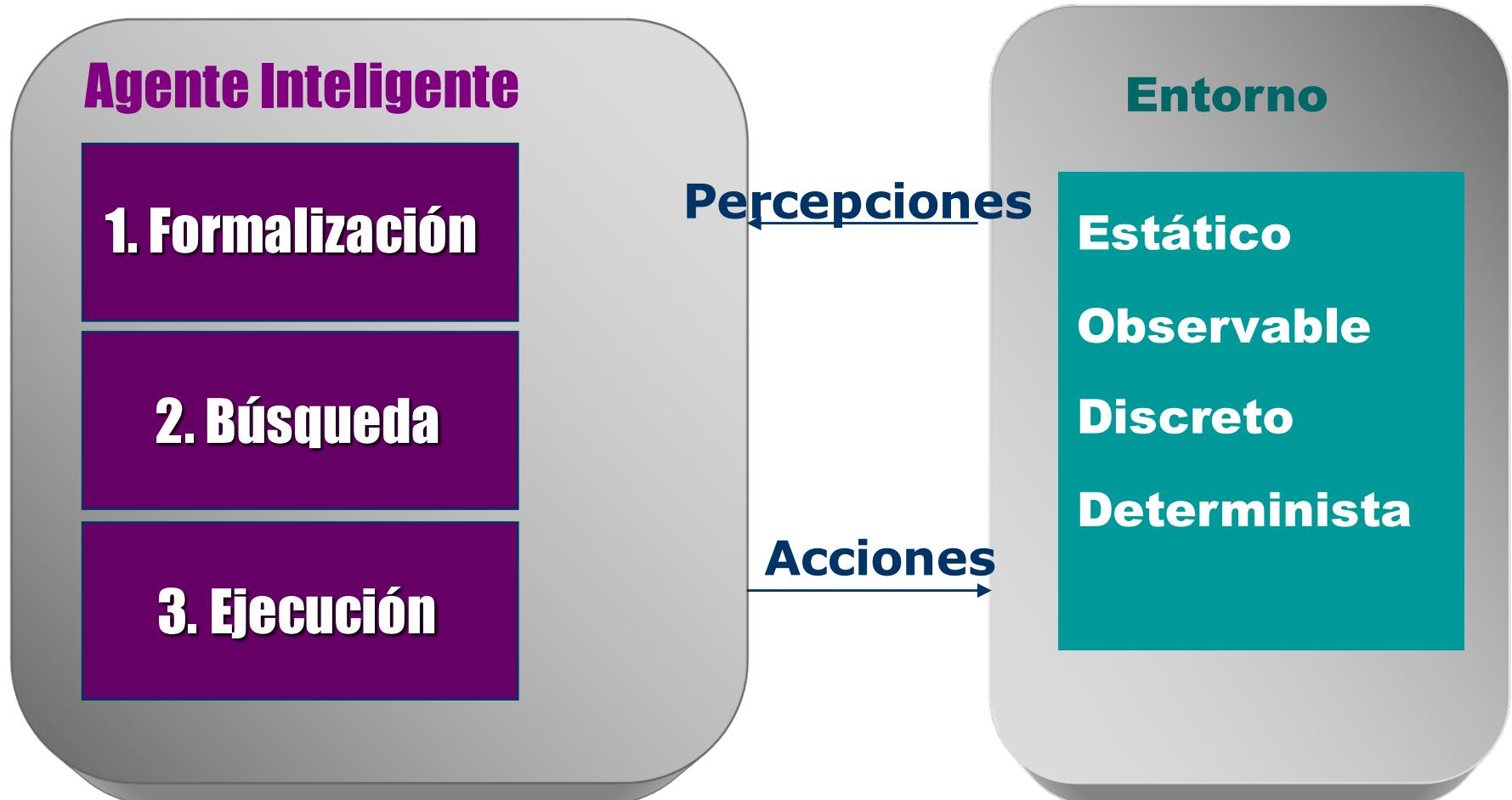


ABSTRACCIÓN

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

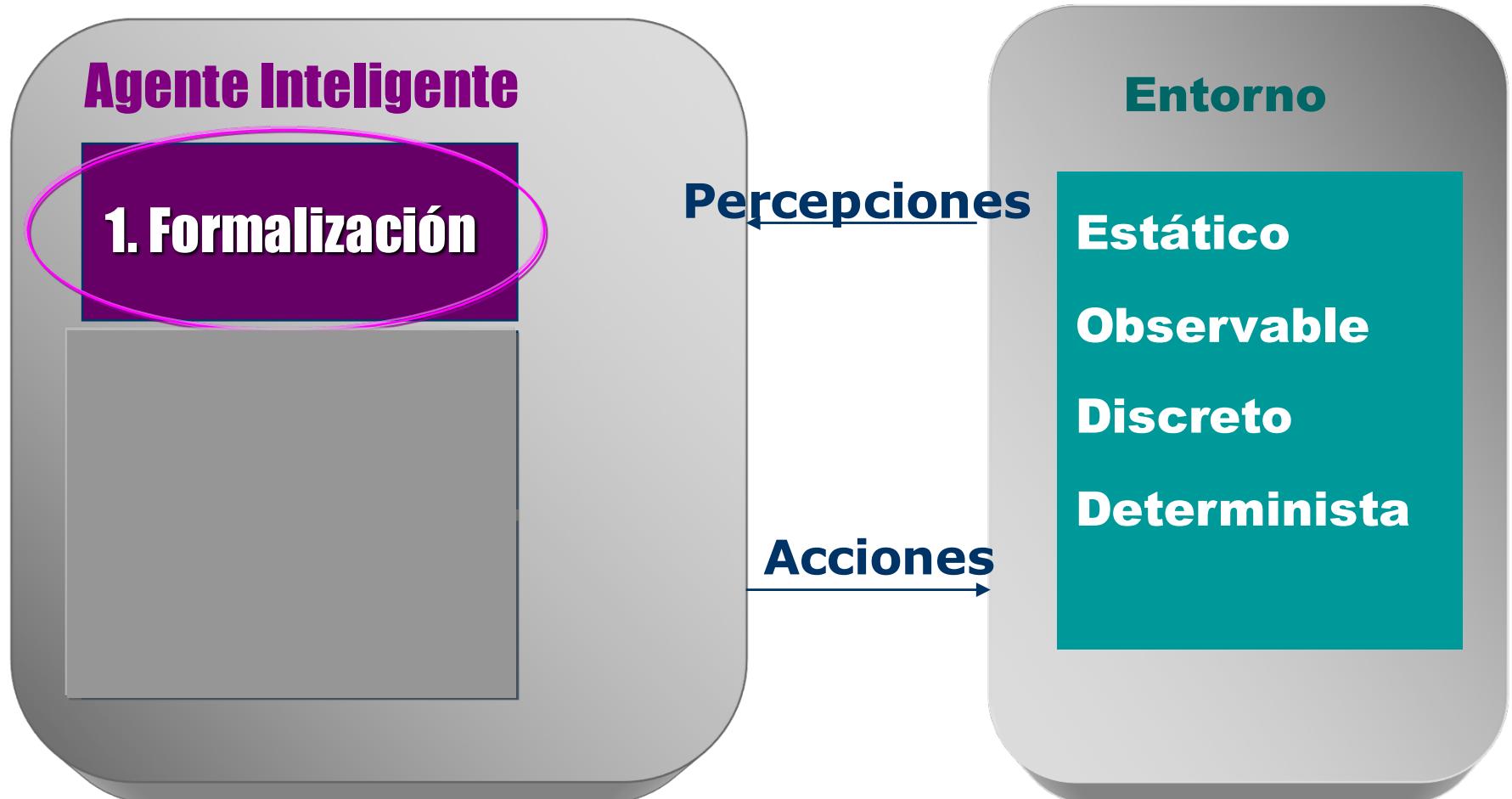
- **Estático:** no se tienen en cuenta los cambios que puedan ocurrir a posteriori
- **Observable:** permite definir el **estado inicial**
- **Discreto:** permite enumerar las **líneas de acción**
- **Determinista**, ya que el **siguiente estado** está totalmente determinado por el estado actual y la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



Inteligencia
Artificial

Departamento de
Ingeniería Informática

Agentes que resuelven problemas



Formalización de un problema de búsqueda

Un problema se define mediante:

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

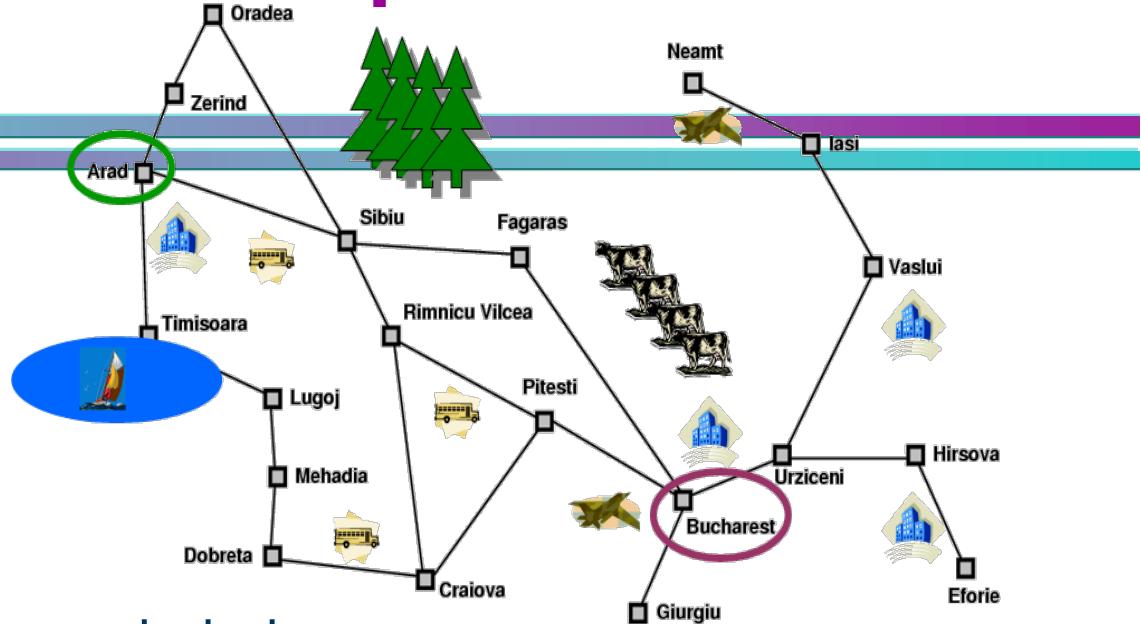
Formalización de un problema de búsqueda

- **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:** que generen los estados sucesores
- **Sucesores**, se generan con las funciones:
 - esValido y aplicaOperador: 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
- **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 de un problema de búsqueda



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

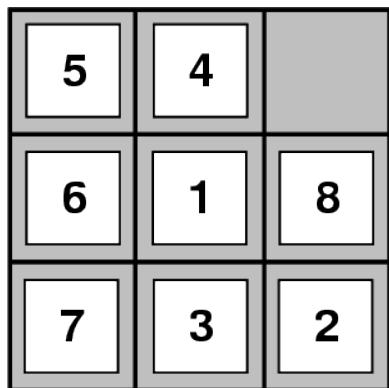
- Estado Inicial
- Test Objetivo
- Lista de operadores
- Sucesores
- Camino o solución
- Coste

Formalización del problema

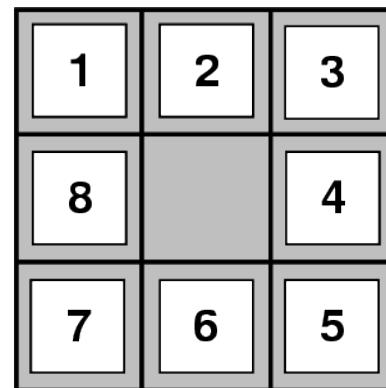
OBJETIVO: Ir a Bucarest

- **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 las 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



Start State



Goal State

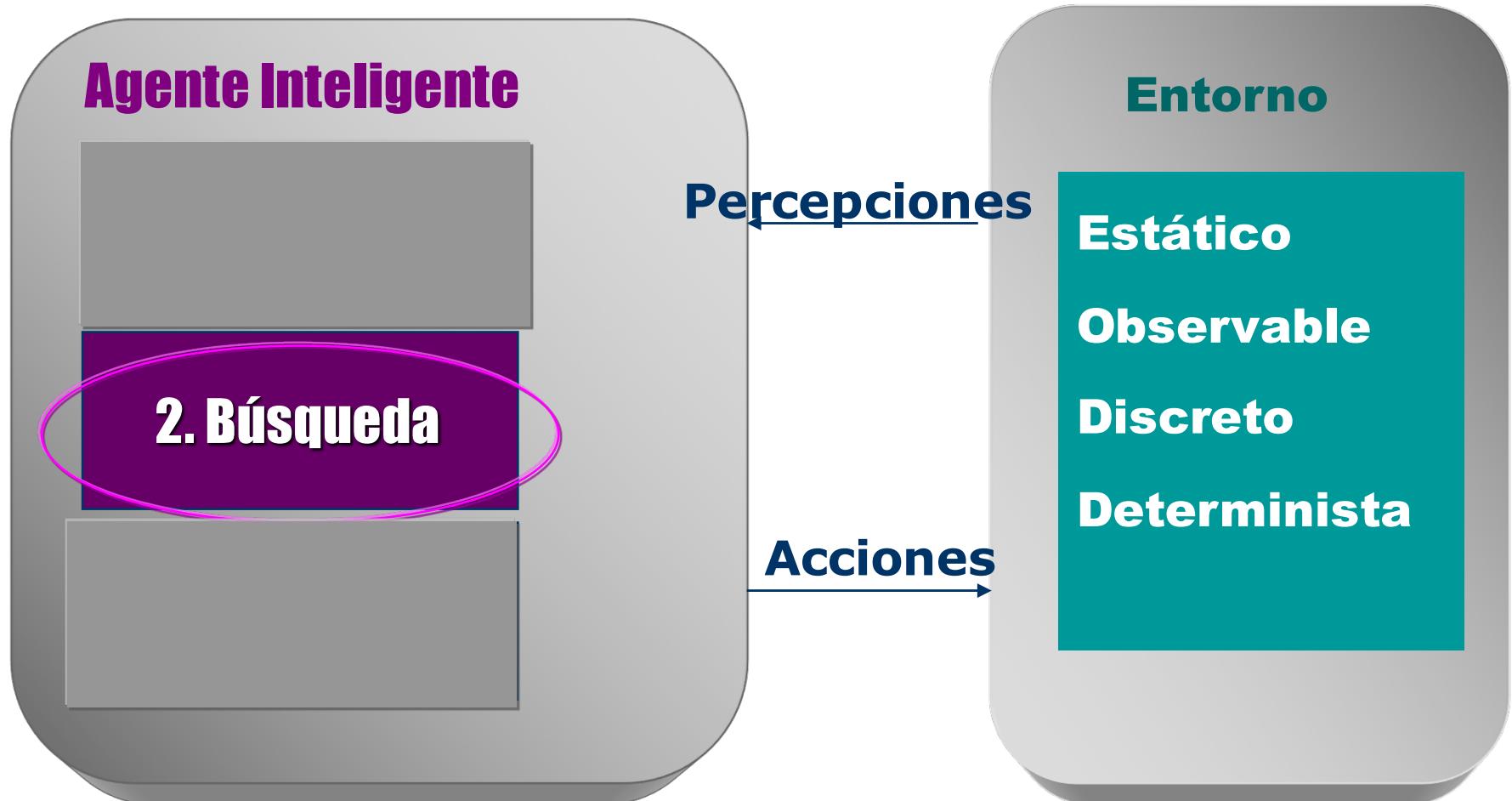
- Estado Inicial
- Test Objetivo
- Lista de operadores
- Sucesores
- Camino o solución
- Coste

Formalización del 8-puzzle

- **Estados:** Tableros o puzzles 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 los 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:** intercambiar 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

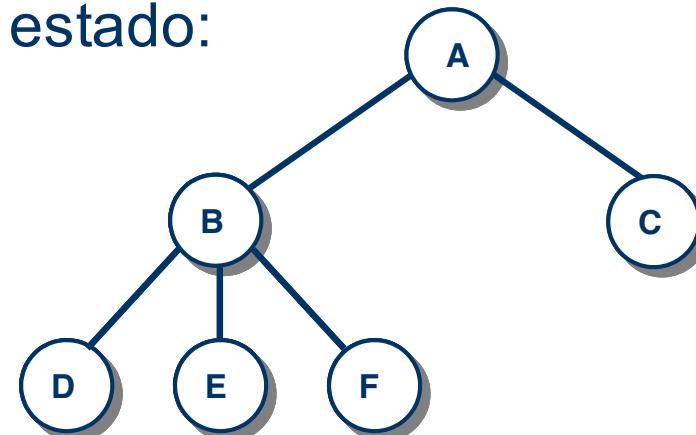
Agentes que resuelven problemas

- **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

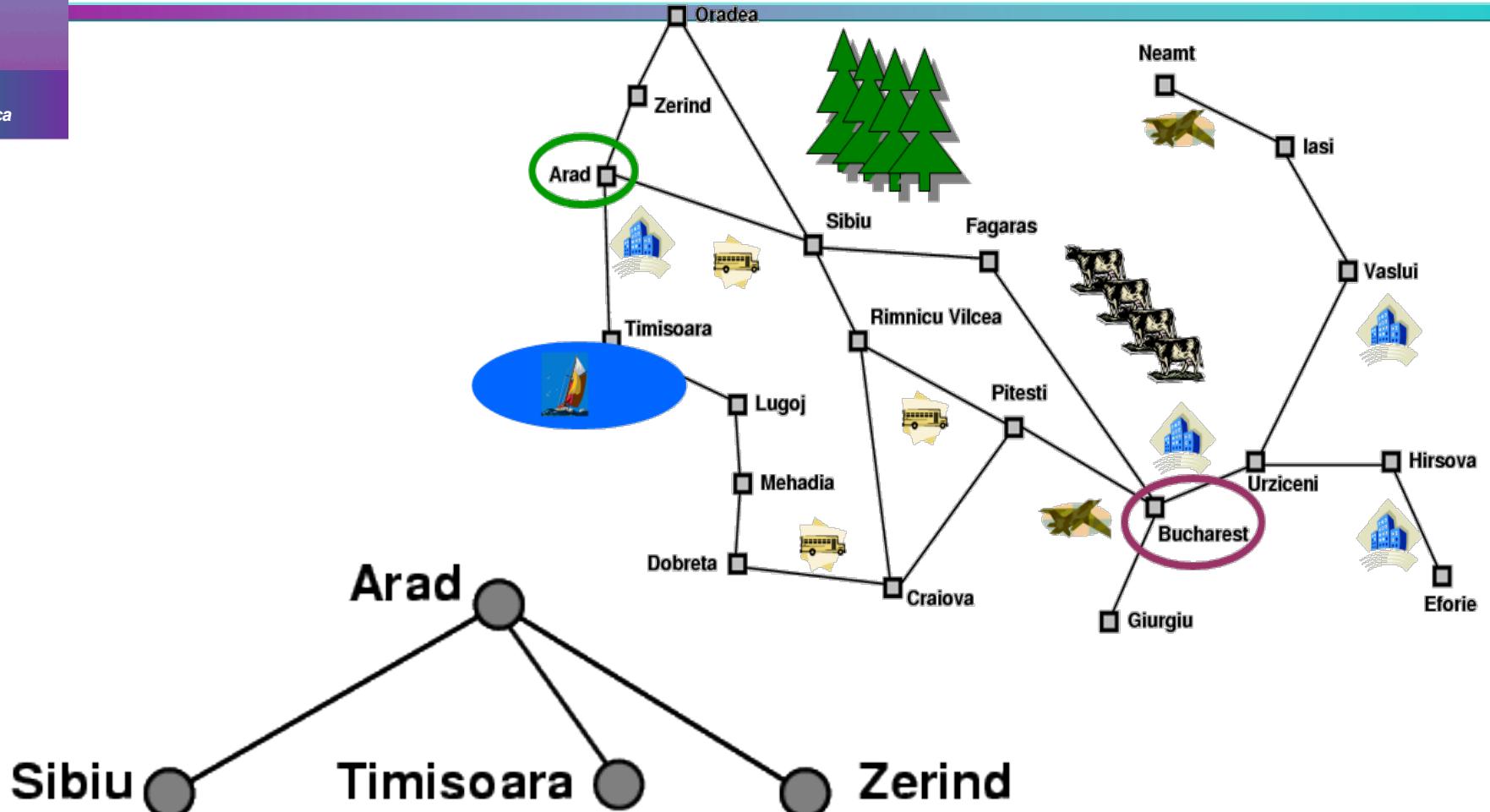


Ejemplo de Expansión

Arad



Ejemplo de Expansión



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



Inteligencia
Artificial

Departamento de
Ingeniería Informática

Funcionamiento básico de la búsqueda

Abiertos=Inicial

Mientras No_Objetivo(Actual) & NoVacia(Abiertos)

 Actual = Primero(Abiertos)

 Sucesores = **Expandir**(Actual)

 Abiertos = Abiertos + Sucesores

Fin_Mientras

Algoritmo General de Búsqueda

Solucion: función Búsqueda (tNodo: Inicial, entero: estrategia)

inicio

 tNodo Actual

 tLista: Abiertos $\leftarrow \{\text{Inicial}\}$ // El nodo inicial se guarda en Abiertos

 logico Objetivo: Falso

mientras (No Vacia(Abiertos)) **Y** (No Objetivo)

 Actual \leftarrow Primero(Abiertos) // selecciona primer nodo de Abiertos

si EsObjetivo(Actual) **entonces**

 Objetivo \leftarrow Verdadero

si_no

 Sucesores \leftarrow Expandir(Actual)

 Abiertos $\leftarrow \{\text{Abiertos} + \text{Sucesores}\}$ //de acuerdo a estrategia

fin_si

 Cerrados $\leftarrow \{\text{Cerrados} + \text{Actual}\}$

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 $\leftarrow \{ \}$

 desde op $\leftarrow 1$ hasta NUM_OPERADORES hacer

 si esValido(op,actual) entonces

 nuevo \leftarrow aplicaOperador(op,actual)

 Sucesores $\leftarrow \{ \text{Sucesores} + \text{nuevo} \}$

 fin_si

fin_desde

devolver Sucesores

fin_función

Medidas del Rendimiento

Completa: la estrategia siempre que exista, encontrará una solución

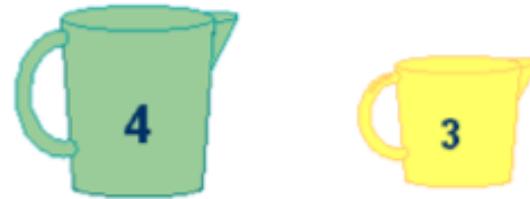
Óptima: la estrategia siempre que exista solución, encontrará primero la mejor solución

Complejidad en tiempo: número de nodos generados durante la búsqueda

Complejidad en espacio: máximo número de nodos en memoria

El problema de la Jarras de Vino

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



- **Salida:** la jarra de 4 litros contiene exactamente 2 litros



- **Operaciones permitidas:**
 - **Llenar** las jarras del depósito
 - **Vaciar** las jarras en el depósito
 - **Pasar** contenido de una jarra a otra hasta que una se vacíe o se llene la otra

- **Medios:** depósito con vino suficiente



Formalización de un problema de búsqueda

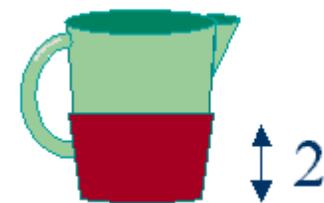
- **Estados:** contenido de cada jarra (variables de tipo entero)
- **Estado Inicial:** Jarras Vacías
- **Test Objetivo:** La jarra de 4 litros tiene exactamente 2 litros (el contenido de la jarra de 3 litros es indiferente)
- **Lista de Operadores:** operaciones de llenar, pasar y vaciar (se enumeraran 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)

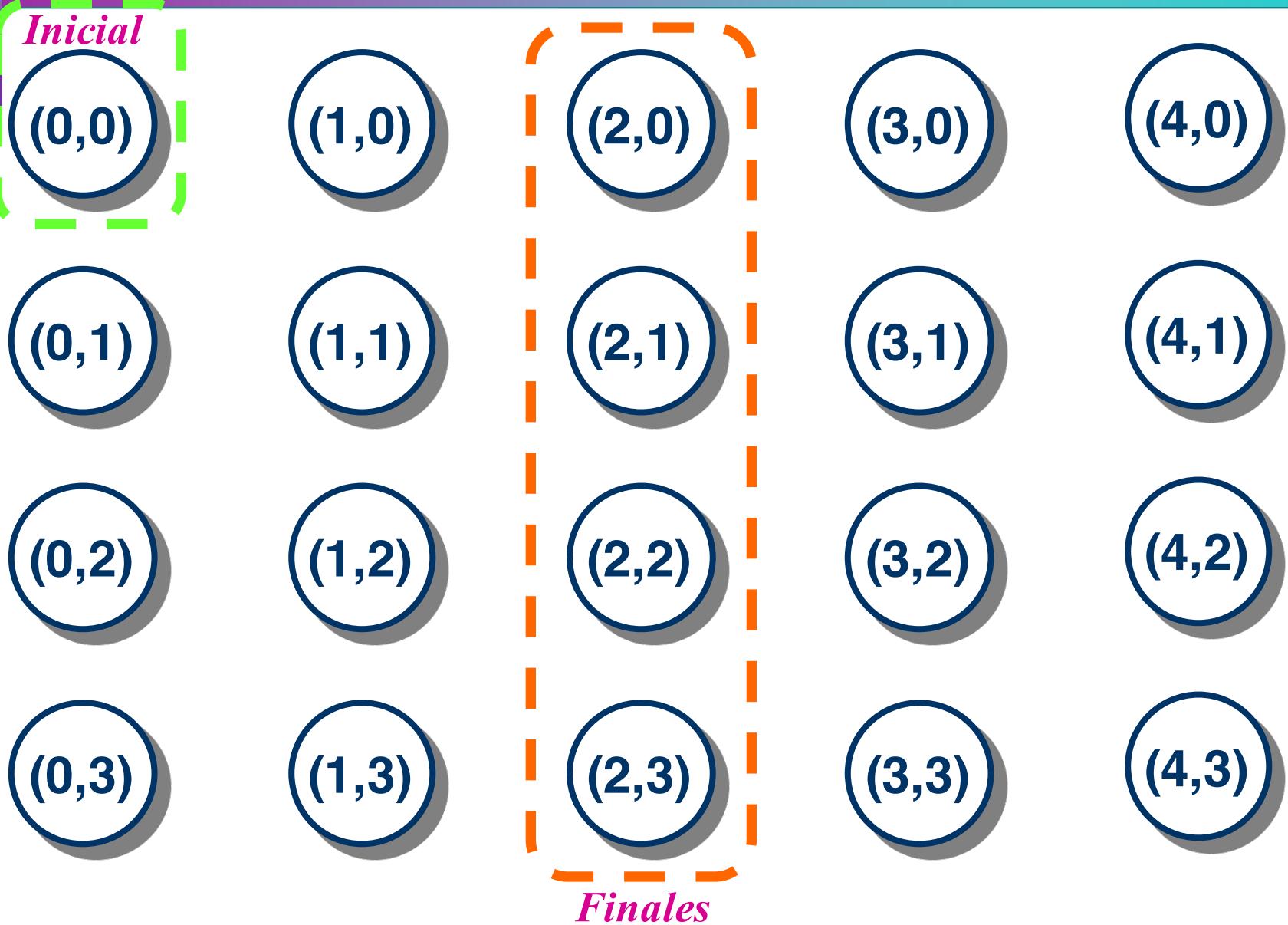


- Espacio de Estados = (x, y)
tal que $x \in \{0,1,2,3,4\}$, $y \in \{0,1,2,3\}$

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



Espacio de Estados (jarras)



Operadores (jarras)

- Llenar J4 $(x,y) \rightarrow (4,y)$
- Llenar J3 $(x,y) \rightarrow (x,3)$
- Vaciar J4 $(x,y) \rightarrow (0,y)$
- Vaciar J3 $(x,y) \rightarrow (x,0)$

- Pasar de la J3 a J4 hasta que se llene o se vacíe la menor
- Pasar de J4 a J3 hasta que se llene o se vacíe J4

Restricciones (jarras)

- No tiene sentido llenar una jarra que ya está llena

Si $x < 4$, $(x,y) \rightarrow (4,y)$

Si $y < 3$, $(x,y) \rightarrow (x,3)$

- No tiene sentido vaciar una jarra que ya está vacía

Si $x > 0$, $(x,y) \rightarrow (0,y)$

Si $y > 0$, $(x,y) \rightarrow (x,0)$

Operadores

1. Llenar J4 (x,y), $x < 4 \rightarrow (4,y)$
2. Llenar J3 (x,y), $y < 3 \rightarrow (x,3)$
3. Vaciar J4 (x,y), $x > 0 \rightarrow (0,y)$
4. Vaciar J3 (x,y), $y > 0 \rightarrow (x,0)$

- Pasar de la J3 a J4 hasta que se llene o se vacíe la menor
- Pasar de J4 a J3 hasta que se llene o se vacíe J4



Inteligencia
Artificial

Departamento de
Ingeniería Informática

Formalización



Operadores

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

Operadores

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

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, ...)

Implementación

■ Jarras.h

Debe contener al menos:

- lista de operadores, definidos como constantes
- constante NUMOPERADORES
- definición del tipo **tEstado**
- prototipos de las funciones exportables:
esValido, aplicaOperador, dispOperador, etc.

■ Jarras.c

- **Funciones imprescindibles:** testObjetivo,
aplicaOperador, esValido, estadoinicial
- **Funciones auxiliares:** dispOperador,
dispEstado, etc.

Jarras.h

```
#define LLENAR4 0
#define LLENAR3 1
#define VACIAR4 2
#define VACIAR3 3
#define LLENAR4CON3 4
#define LLENAR3CON4 5
#define VACIAR3EN4 6
#define VACIAR4EN3 7
#define NUM_OPERADORES 8
```

```
typedef struct tEstado {
    int J4, J3;
} tEstado;
```

```
tEstado *estadolInicial();  
  
int testObjetivo(tEstado *estado);  
int esValido(unsigned op, tEstado*s);  
tEstado *aplicaOperador(unsigned op,tEstado *s);
```

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

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



Inteligencia
Artificial

Departamento de
Ingeniería Informática

Jarras.c

```
tEstado *estadolInicial(){  
    tEstado *estado = (tEstado *) malloc(sizeof(tEstado));  
    estado->J4=0;  
    estado->J3=0;  
    return estado;  
} //estadolInicial
```

```
int testObjetivo(tEstado *estado){  
    return(estado->J4==2);  
} //testObjetivo
```

Jarras.c

```
// restricciones para realizar el movimiento
int esValido(unsigned op, tEstado *s){
    int valido;
    switch(op){
        case LLENAR4: valido= s->J4==0;
                        break;
        case LLENAR3: valido= s->J3==0;
                        break;
        case VACIAR4: valido= s->J4>0; break;
        case VACIAR3: valido = s->J3>0;break;
        (.....)
        default: valido=0;
        break; }
    return valido;
} //esValido
```



Inteligencia
Artificial

Departamento de
Ingeniería Informática

Jarras.c

```
tEstado *aplicaOperador(unsigned op, tEstado *a){  
    tEstado *s = (tEstado *) malloc(sizeof(tEstado));  
    s->J4=a->J4;  
    s->J3=a->J3;  
    switch(op) {  
        case LLENAR4:  s->J4=4; break;  
        case LLENAR3:  s->J3=3; break;  
        (...)  
        default: break;  
    }  
    return s;  
} //aplicaOperador
```



Inteligencia
Artificial

Departamento de
Ingeniería Informática

Jarras.c

```
void dispEstado(tEstado *s){  
    printf("\n J4=%d  J3=%d \n", s->J4, s->J3);  
} //dispEstado
```

```
void dispOperador(unsigned op){  
    switch(op) {  
        case LLENAR4: printf("\n Llenar4\n"); break;  
        default: printf("\n No Operador \n"); break;  
    } //dispOperador
```



Inteligencia
Artificial

Departamento de
Ingeniería Informática

Prueba del código: main.c

```
#include ...  
  
int main() {  
    //pruebas del código generado  
    int op;  
    tEstado n,a;  
    a=estadolInicial();  
  
    for (op=1; op<=NUM_OPERADORES; op++) {  
        if (esValido(op, a)){  
            n=aplicaOperador(op,estado);  
            dispOperador(op);  
            dispEstado(n);  
        } //if  
    } //for  
} //main
```