



# Introducción a la Inteligencia Artificial

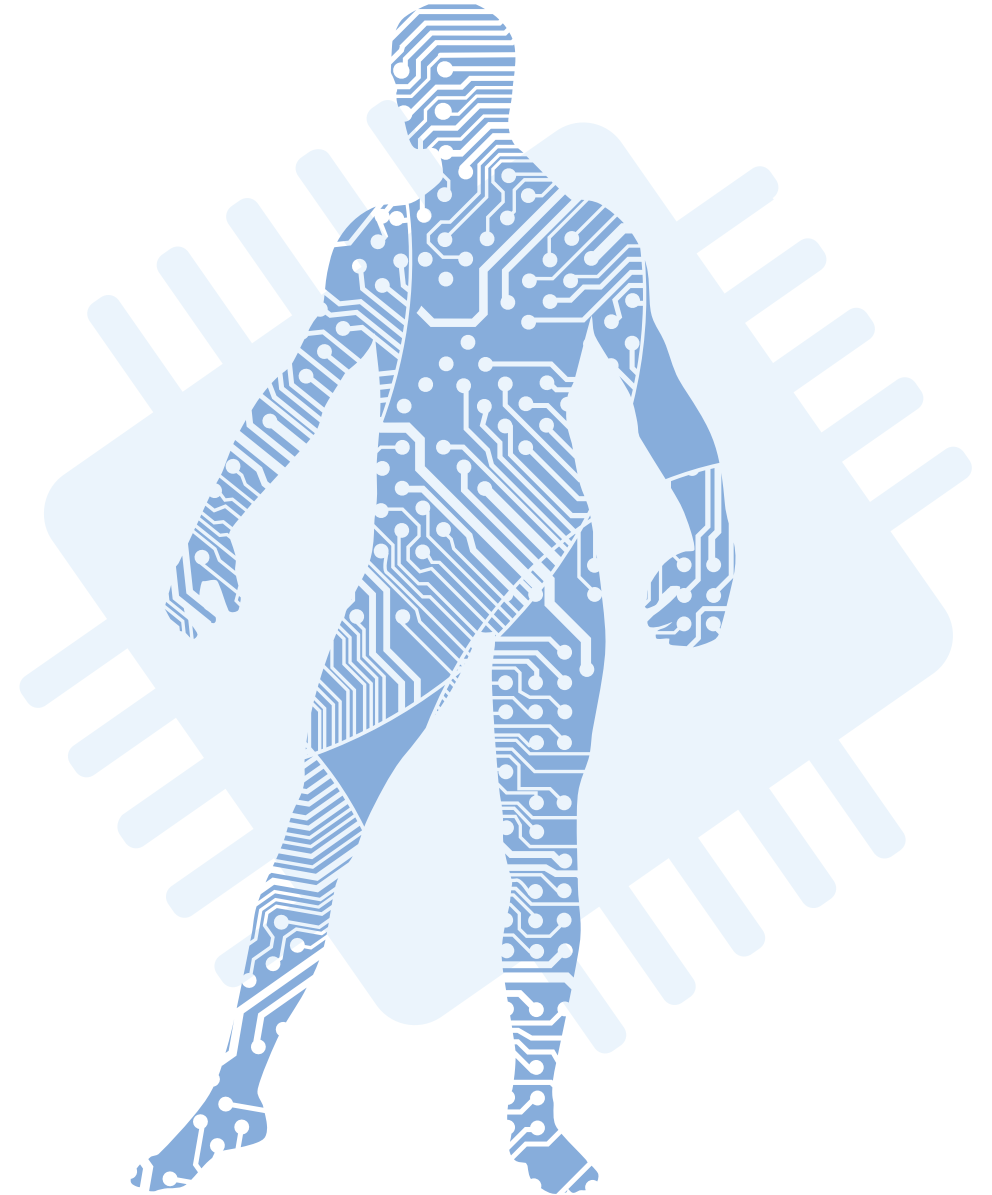
Maestría en Ciencias de la Computación  
Postgrado del Decanato de Ciencias y Tecnología

Dra. Ma. Auxiliadora Pérez

# Agenda

**01** Estrategias de Búsqueda Sistémicas

**02** Estrategias de Búsqueda Heurísticas



**El problema de las jarras de agua:** “Se tienen dos jarras sin marcas de medición y con capacidades 4 y 3 litros. Asumiendo que se tiene agua suficiente, cómo medir exacta-mente 2 litros en la jarra con capacidad 4?”

**Representación:**

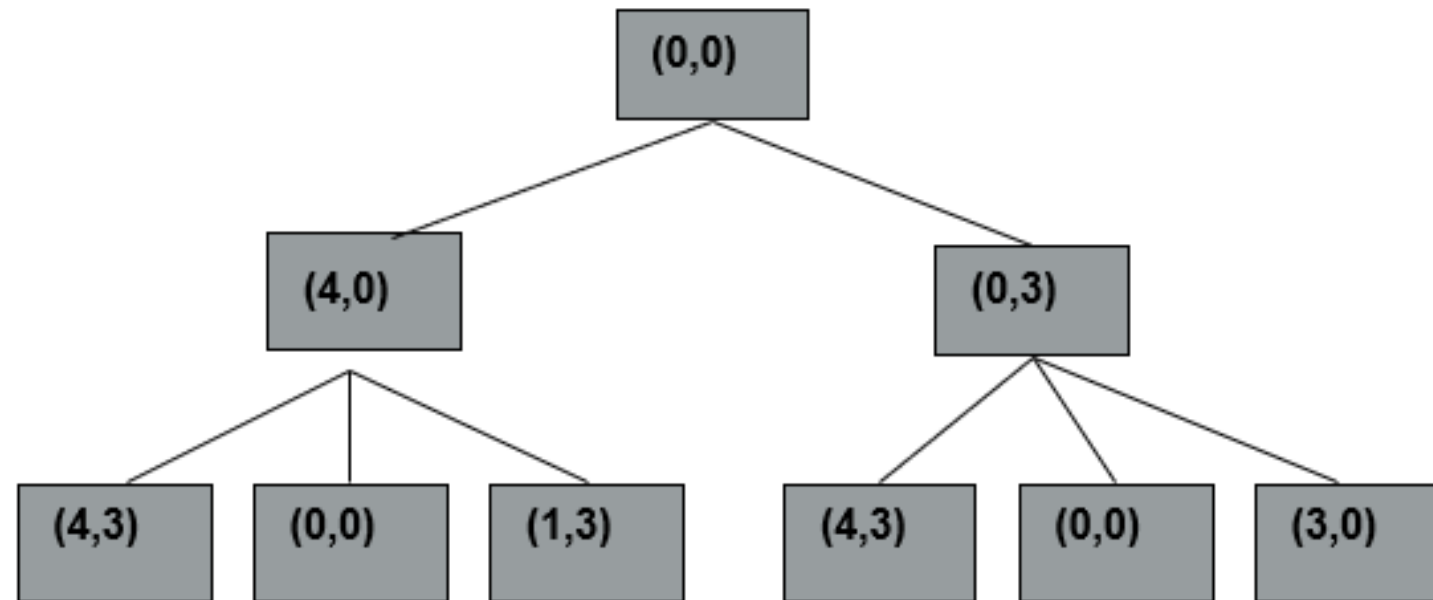
*Espacio de Estados:* Pares ordenados  $(x,y)$   
 $(c4,c3)$

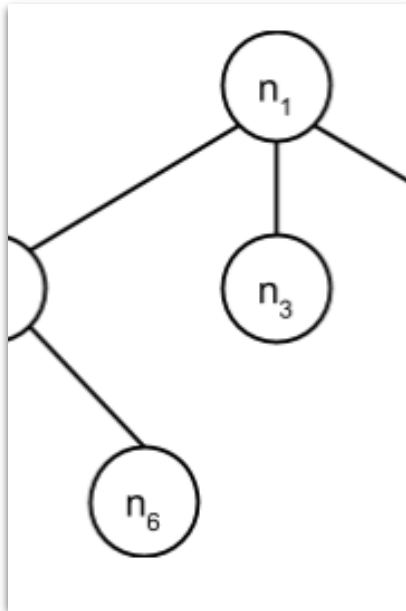
*Estado inicial:*  $(0,0)$

*Estado final:*  $(2,n)$



# Problema de Búsqueda





Dado un estado inicial y varios posibles estados sucesores

- **Estrategias sistemáticas:**  
No tienen preferencias entre ellos. Difieren entre sí en el orden de expansión de nodos.
- **Estrategias heurísticas :**  
Hacen uso del conocimiento del dominio para elegir entre ellos, generando mayor efectividad en la búsqueda.



# Búsqueda a ciegas

Métodos que no usan ningún conocimiento específico acerca del problema

Primero en profundidad

Primero en amplitud

Búsqueda no-determinística

Profundización Iterativa

Búsqueda bidireccional

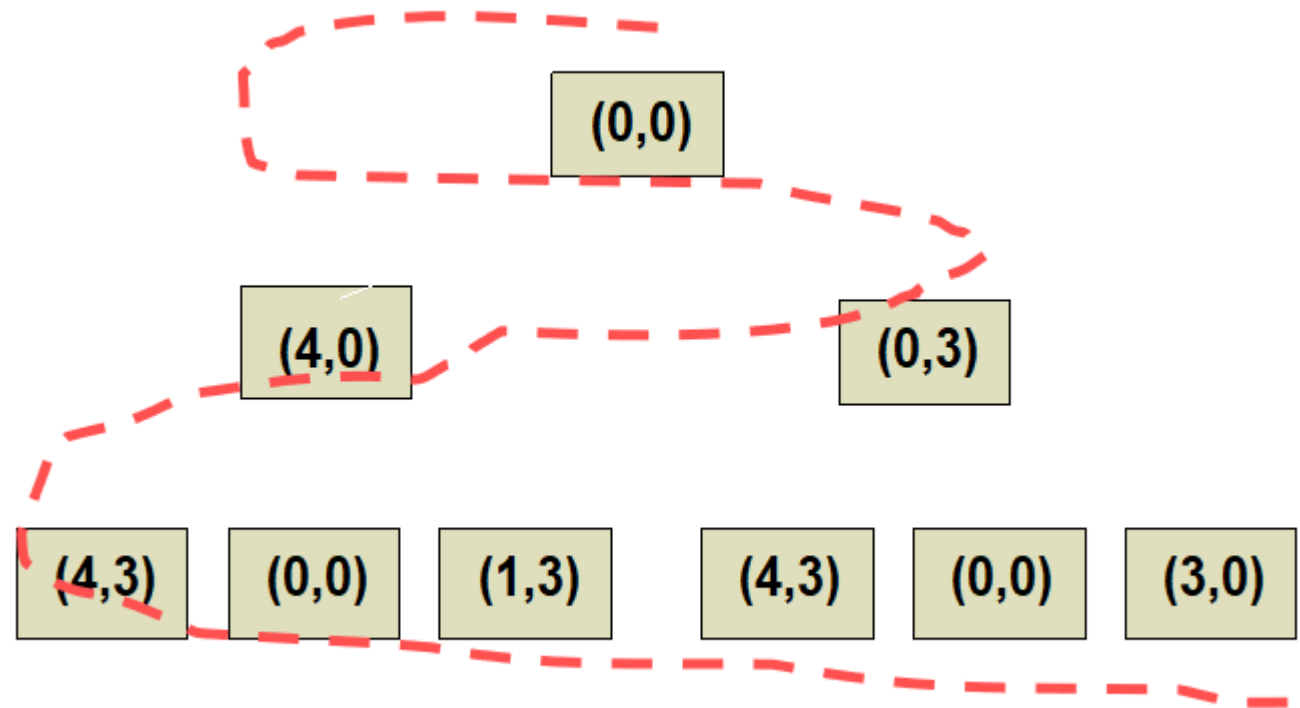
- **Orden de expansión de nodos:**

Nodo raíz , sus sucesores (nivel d),  
los sucesores de ellos (nivel d + 1)

- **Algoritmo:** general con inserción posterior en la cola

**Algoritmo:**

1. Inserte en una cola el elemento raíz (nodo de partida)
2. Hasta que el elemento frontal sea el nodo meta, o se vacíe la cola
  - a) Si nodo frontal tiene hijos, insertar todos sus hijos al final de la cola.
  - b) Eliminar nodo frontal.
3. Si el nodo meta se alcanza, retorne true, de lo contrario, retorne false.





## Evaluación de Primero en Amplitud (BFS)

- Completitud: si  $r$  es finito
- Complejidad en tiempo:  $O(r^p)$
- Complejidad en espacio:  $O(r^p)$
- Optimalidad: siempre que el costo sea no-decreciente

Un algoritmo es completo si, dado un conjunto de posibles soluciones (espacio de búsqueda) de tamaño finito ' $r$ ', el algoritmo es capaz de encontrar una solución si existe, o determinar que no existe ninguna solución.

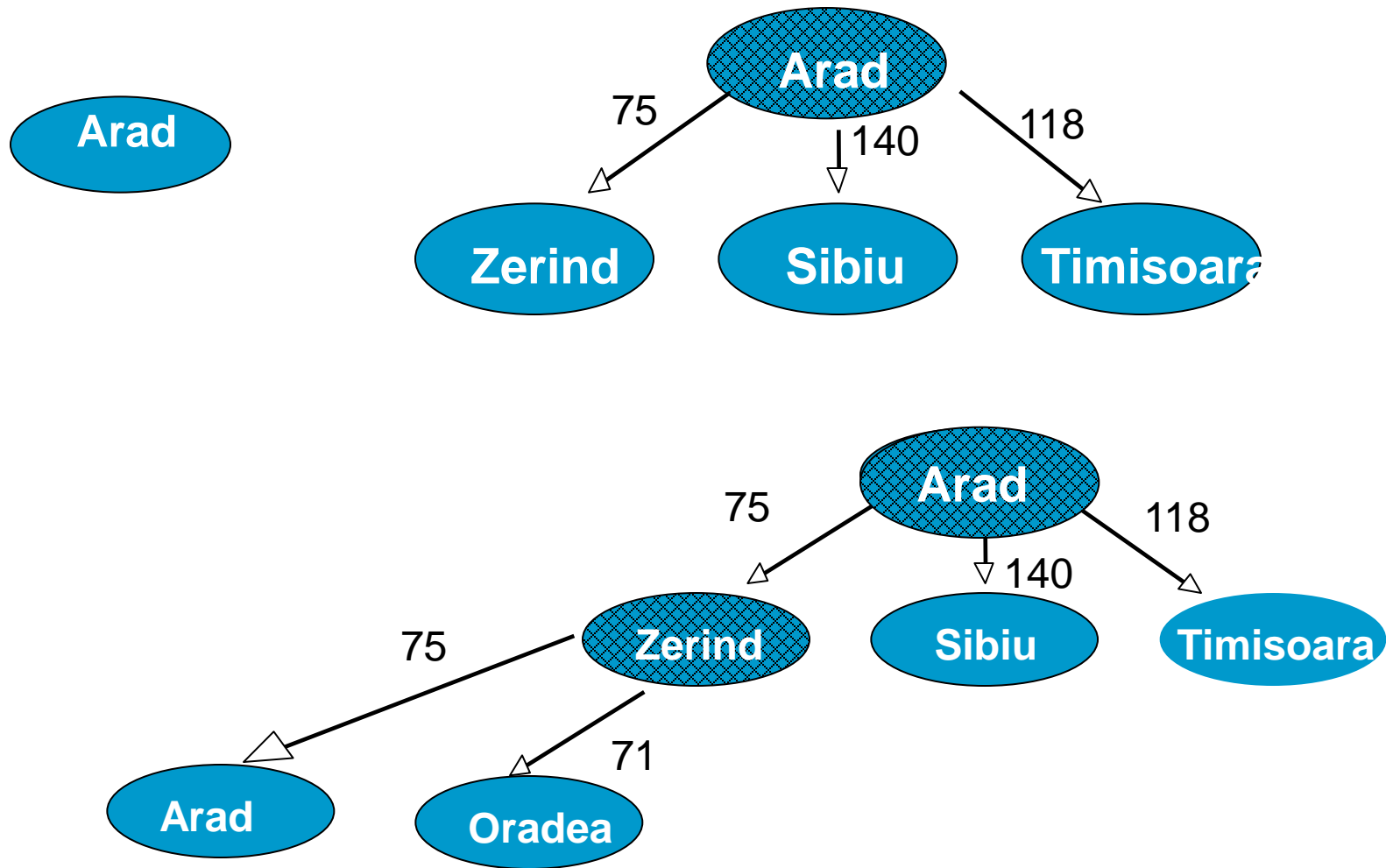
La complejidad en tiempo de un algoritmo describe cómo crece el tiempo de ejecución del algoritmo en función del tamaño de la entrada (' $r$ ' en este caso, que representa el tamaño del espacio de búsqueda, y ' $p$ ' que podría representar otro parámetro relevante).

La complejidad en espacio se refiere a cuánto espacio de memoria (o almacenamiento) necesita un algoritmo para ejecutar su tarea en función del tamaño de la entrada.

Un algoritmo se considera óptimo si resuelve un problema utilizando la menor cantidad de recursos posibles. En este caso la optimalidad está condicionada a que el costo (o la función objetivo que se busca minimizar) sea no-decreciente. Esto significa que a medida que se exploran más soluciones, el costo nunca disminuye.

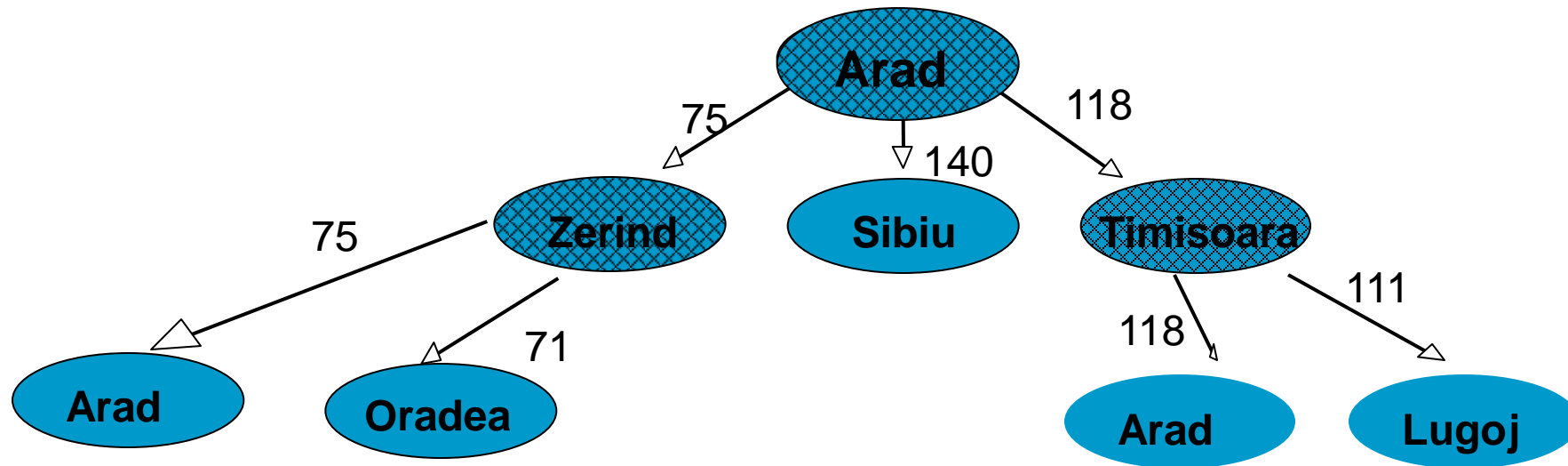


# A Estrategias Sistemáticas: Costo Uniforme







# Estrategias Sistemáticas: Costo Uniforme





## Evaluación de Costo Uniforme

- Completitud  si costo no decrece
- Complejidad en tiempo  $O(r^p)$
- Complejidad en espacio  $O(r^p)$
- Optimalidad 

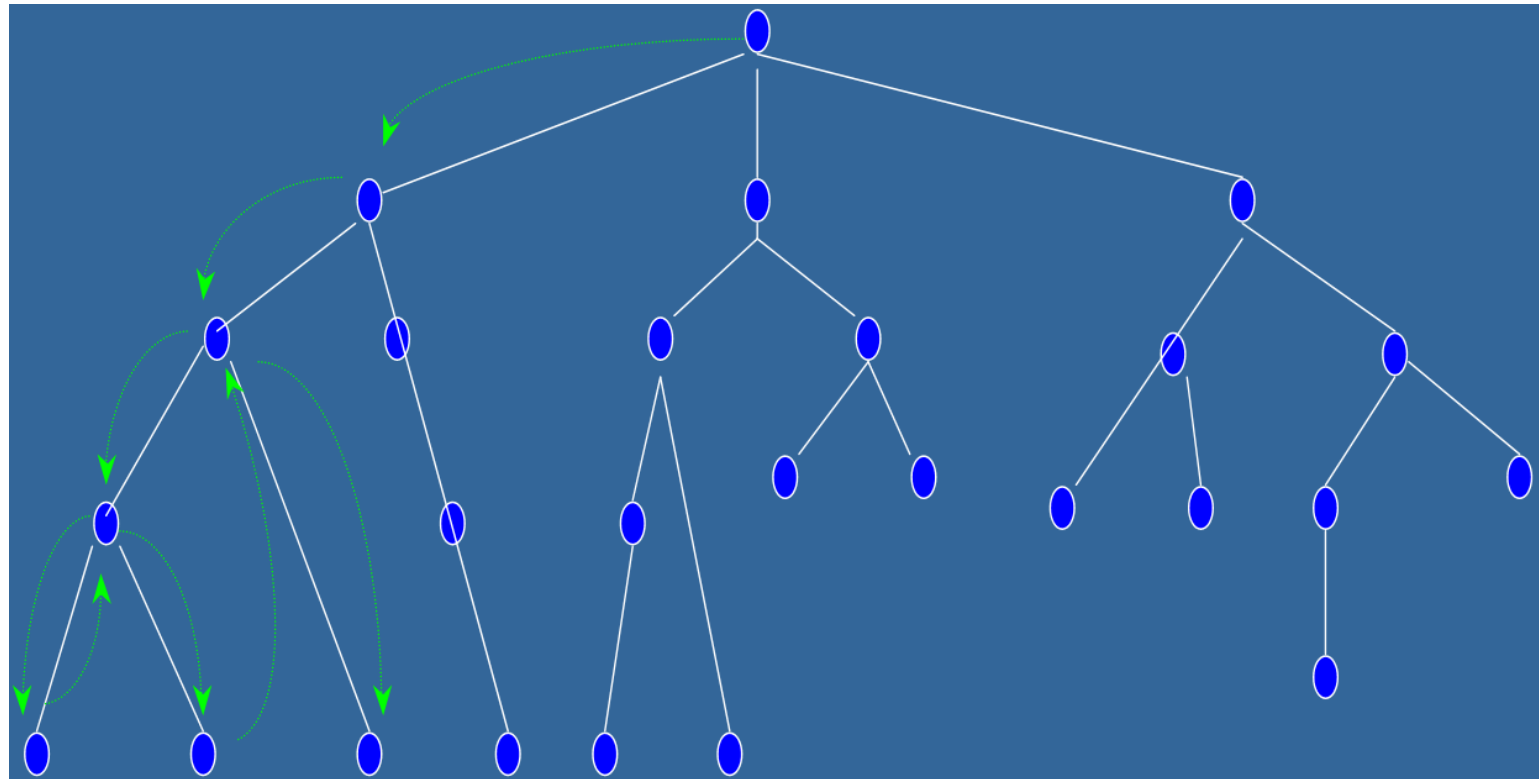
# A Estrategias Sistemáticas: Primero en Profundidad (DFS)

## Algoritmo:

1. Inserte en una pila el elemento raíz (nodo de partida)
2. Hasta que el elemento tope sea el nodo meta, o se vacíe la pila
3. Si nodo tope tiene hijos, insertar el hijo siguiente aun no visitado, según ordenamiento.
4. Si no, entonces eliminar nodo tope.
5. Si el nodo meta se alcanza, retorne true, de lo contrario, retorne false.

## Problemas:

- Puede caer en ciclos infinitos
- Requiere de espacio de búsqueda finito, no-cíclico (o chequear por estados repetidos)





# Evaluación de Primero en Profundidad

- Completitud Sólo si  $r$  es finito
- Complejidad en tiempo  $O(r^m)$
- Complejidad en espacio  $O(r \cdot m)$
- Optimalidad No

- Controla límite de profundidad en nivel de búsqueda: diámetro, ampliándolo iterativamente
- Combina beneficios de primero en amplitud (óptimo y completo) y en profundidad (modestos requerimientos de memoria)
- Expande nodos como en primero en amplitud, pero algunos nodos se expanden varias veces

```
function PROFUNDIZACION-ITERATIVA  
  (problema) returns una secuencia de  
  solución  
  
  inputs: problema, un problema  
  
  for profundidad  $\leftarrow 0$  to  $\infty$  do  
    resultado  $\leftarrow$  PROFUNDIZACION-  
      ITERATIVA (problema, profundidad)  
    if resultado  $\neq$  corte then return resultado  
  end
```



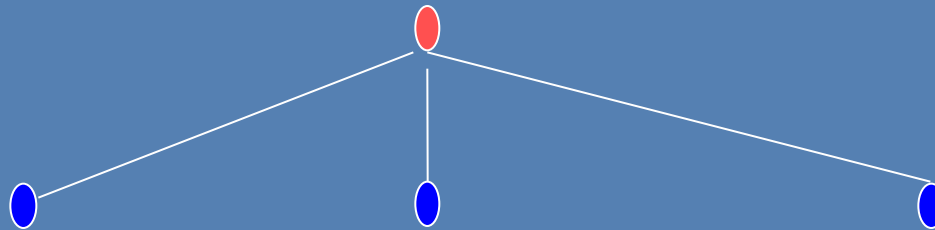
# Estrategias Sistemáticas: Profundización Iterativa

$L = 0$



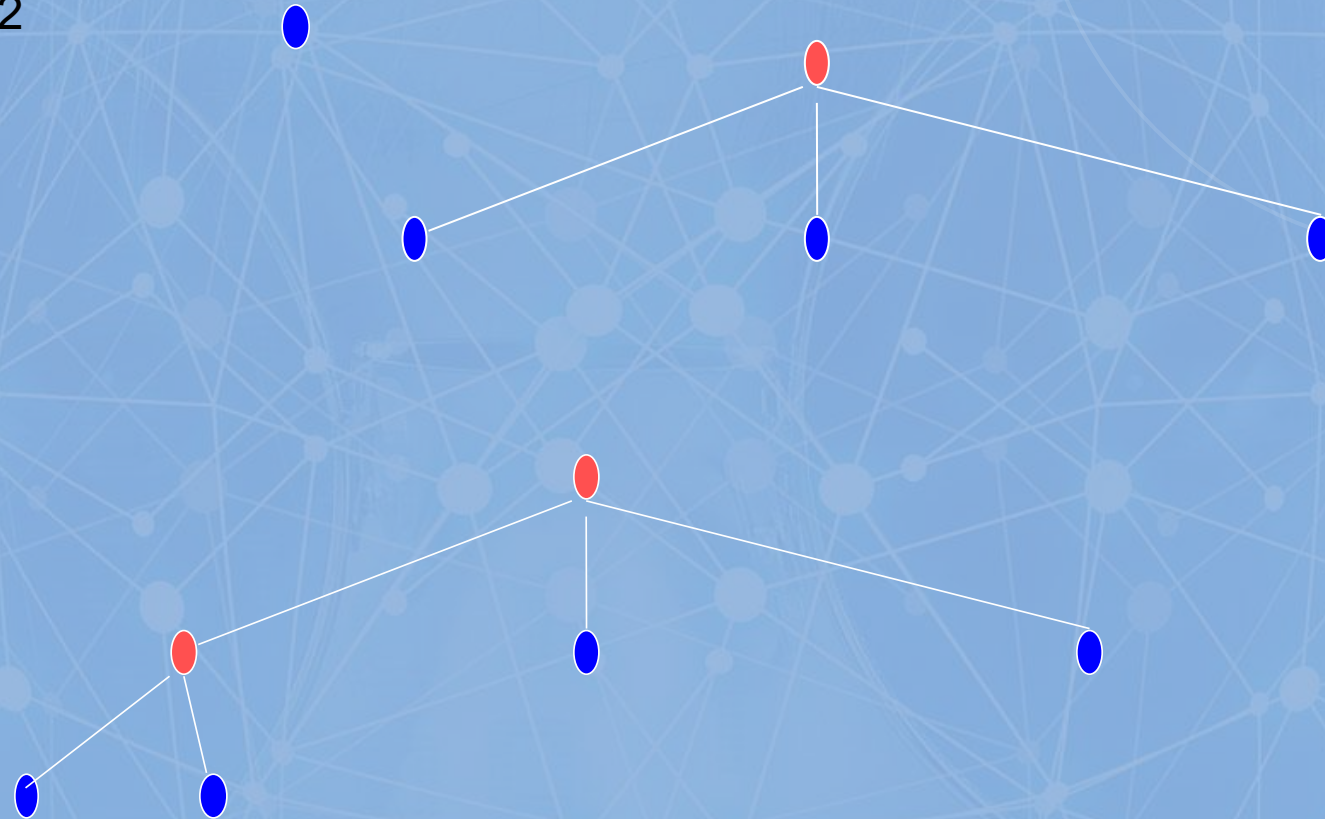
## Estrategias Sistemáticas: Profundización Iterativa

$L = 1$

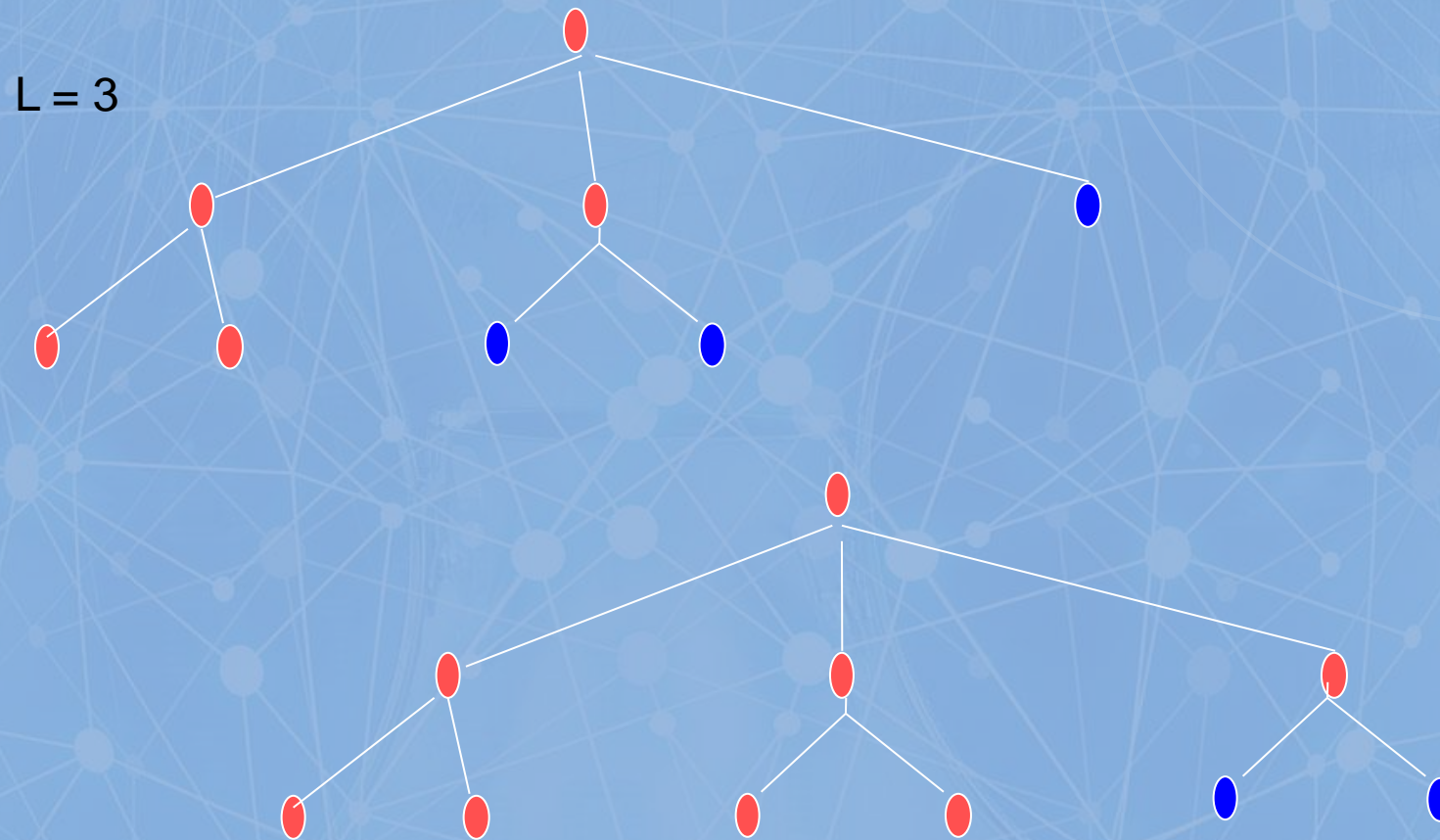


## Estrategias Sistemáticas: Profundización Iterativa

$L = 2$



## Estrategias Sistemáticas: Profundización Iterativa





# Evaluación de Primero en Profundidad

- Completitud: Sí
- Complejidad en tiempo:  $O(r^p)$
- Complejidad en espacio:  $O(r^p)$
- Optimalidad Sí, si costo de  $c/\text{tramo}=1$



# Estrategias Heurísticas



# Estrategias Heurísticas



Cuando se dispone de más información que el estado inicial, los operadores y el test de estado objetivo, el tamaño del espacio de búsqueda usualmente puede reducirse.

A mejor información disponible, más eficientes procesos de búsqueda.

Tales métodos o estrategias son basados en información heurística.

## □ Heurísticas:

(Del v. griego ***heuriskein***: encontrar, descubrir).  
Reglas empíricas o técnicas de juicio que, en algunos casos, conducen a una solución, aunque no garantizan el éxito.

## □ Ejemplos:

- Vendedor Viajero: Dist. a estado objetivo
- 8-puzzle: N<sup>o</sup> fichas fuera de sitio o Manhattan

Su eficacia depende en gran medida de la forma en que exploten el conocimiento del dominio particular del problema.

Proporcionan un marco donde situar el conocimiento del dominio específico.

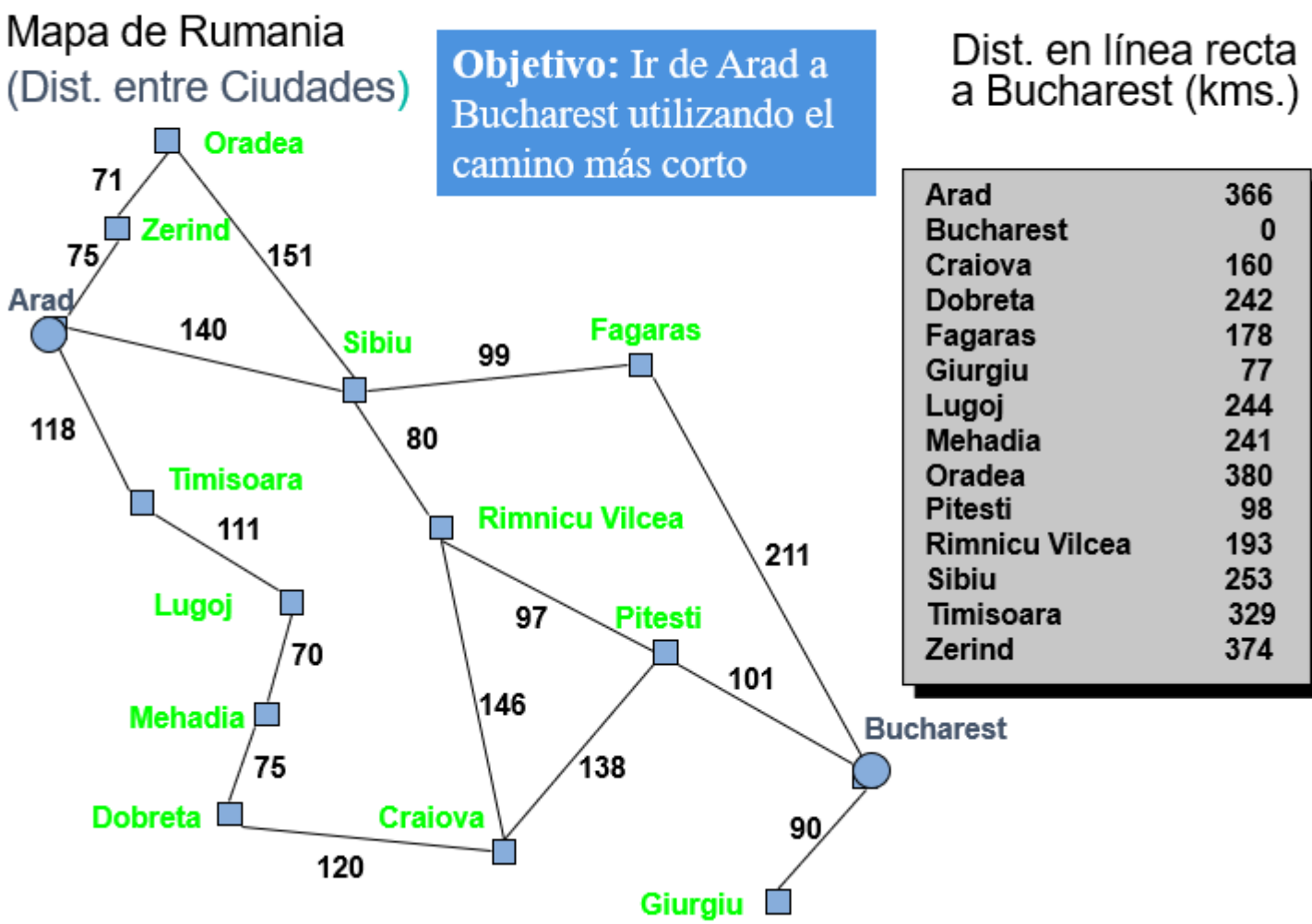
Forman el núcleo de la mayoría de los sistemas de IA.

## Algoritmo General

- Utiliza una función heurística  $h(n)$  para estimar el costo necesario del camino más económico para alcanzar el nodo objetivo a partir del nodo actual.
- Una buena función heurística para problemas de búsqueda de ruta es la **distancia en línea recta al objetivo**.

```
function BUSQUEDA-GENERAL (problema, INSERTA-FN)  
  returns una solución, o falla  
  nodos  $\leftarrow$  HACER-COLA (HACER-NODO  
    (ESTADO-INICIAL[problema]))  
  loop do  
    if nodos está vacío then return falla  
    nodo  $\leftarrow$  REMUEVE-FRENTE (nodos)  
    if test-meta [ problema] aplicado a  
      ESTADO(nodo) tiene éxito  
    then return nodo  
    nodos  $\leftarrow$  INSERTA-FN (nodos, EXPANDE (nodo,  
      OPERADORES[problema]))  
end
```

Costos de recorridos en Rumania en Kms

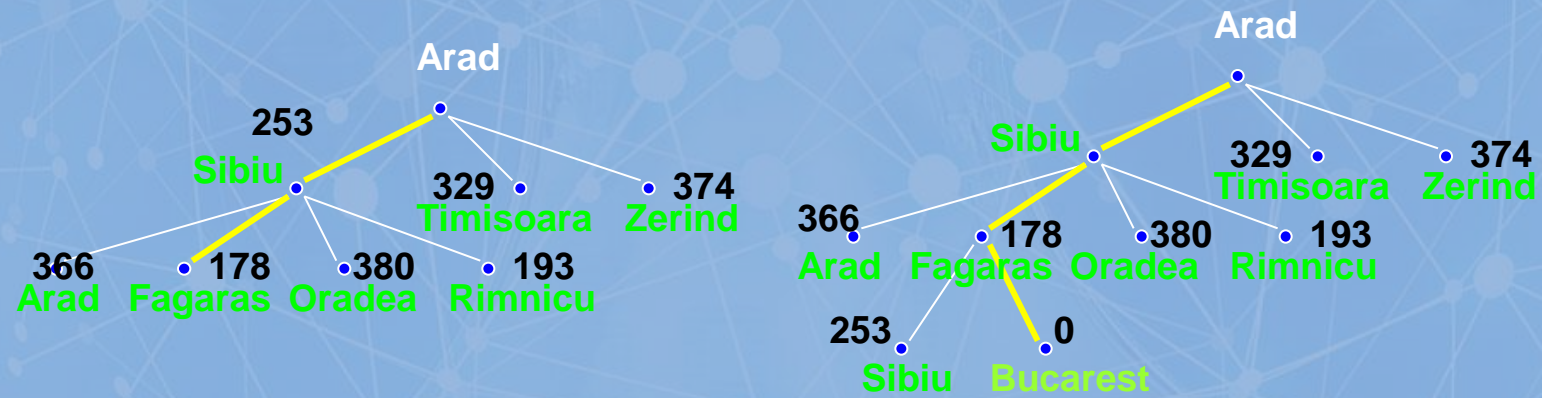


# Estrategias Heurísticas

Función heurística:  
Dist. en línea recta  
al objetivo



Paso 2



Paso 4

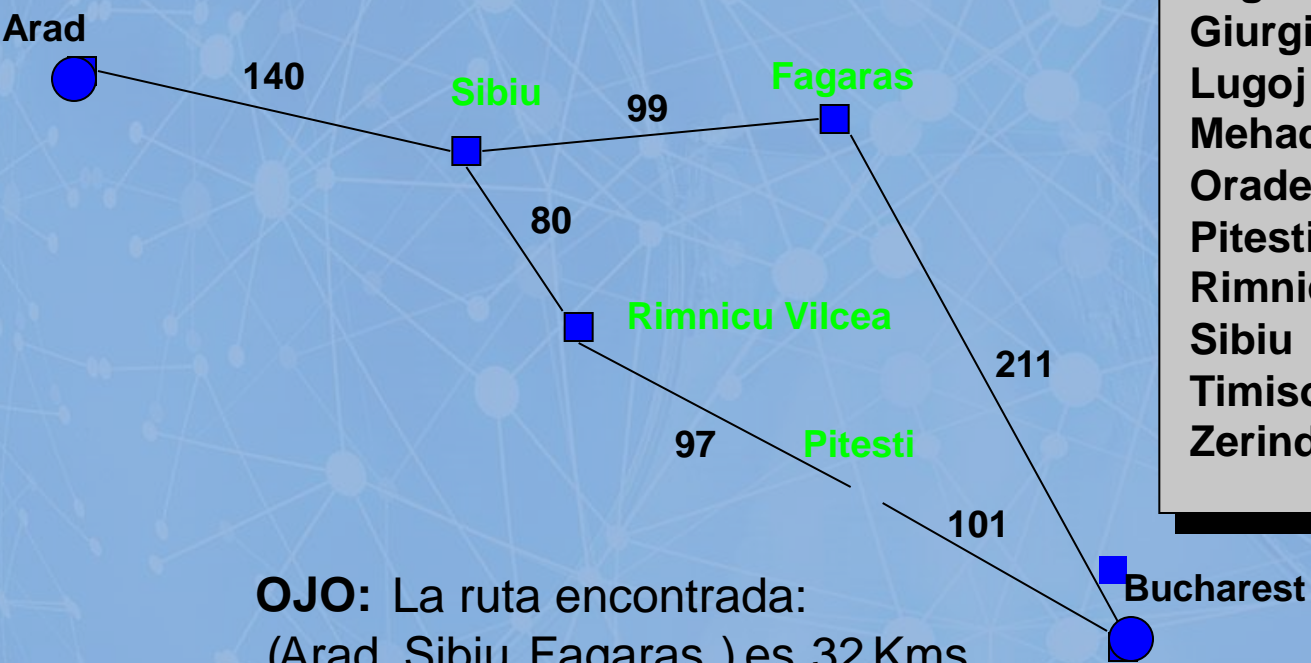
Paso 3

# Estrategias Heurísticas

## Análisis del Resultado

**Objetivo:** Ir de Arad a Bucharest utilizando el camino más corto

Mapa de Rumania  
(Dist. entre Ciudades)



Dist. en línea recta  
a Buchares(kms)

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Fagaras	178
Giurgiu	77
Lugoj	244
Mehadia	241
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Zerind	374

**OJO:** La ruta encontrada:  
(Arad, Sibiu, Fagaras ) es 32 Kms.  
más larga que la ruta:  
(Arad, Sibiu, Rimnicu, Pitesti)



# Evaluación de Primero el Mejor

- **Complejidad:** No, puede quedar atrapado en loops. Ej Giurgiu  $\leftarrow$  Craiova. Sólo si chequea repeticiones
- **Complejidad en tiempo:**  $O(r^m)$
- **Complejidad en espacio:**  $O(r^m)$
- **Optimalidad:** No

# A Estrategias Heurísticas: A Óptima (A\*)

- ❑ Utiliza en su algoritmo una **función de evaluación**: estimación del costo necesario para alcanzar un estado objetivo por el camino que se ha seguido para generar el nodo actual

$$f(n) = h(n) + g(n)$$

**f(n)** representa el costo estimado de la solución más económica a través del nodo **n**

A\* es una estrategia completa y óptima **siempre que la función heurística h(n) sea admisible**; esto es, nunca sobre-estime el costo para alcanzar el objetivo.

Es decir,... sea **optimista !!!!**

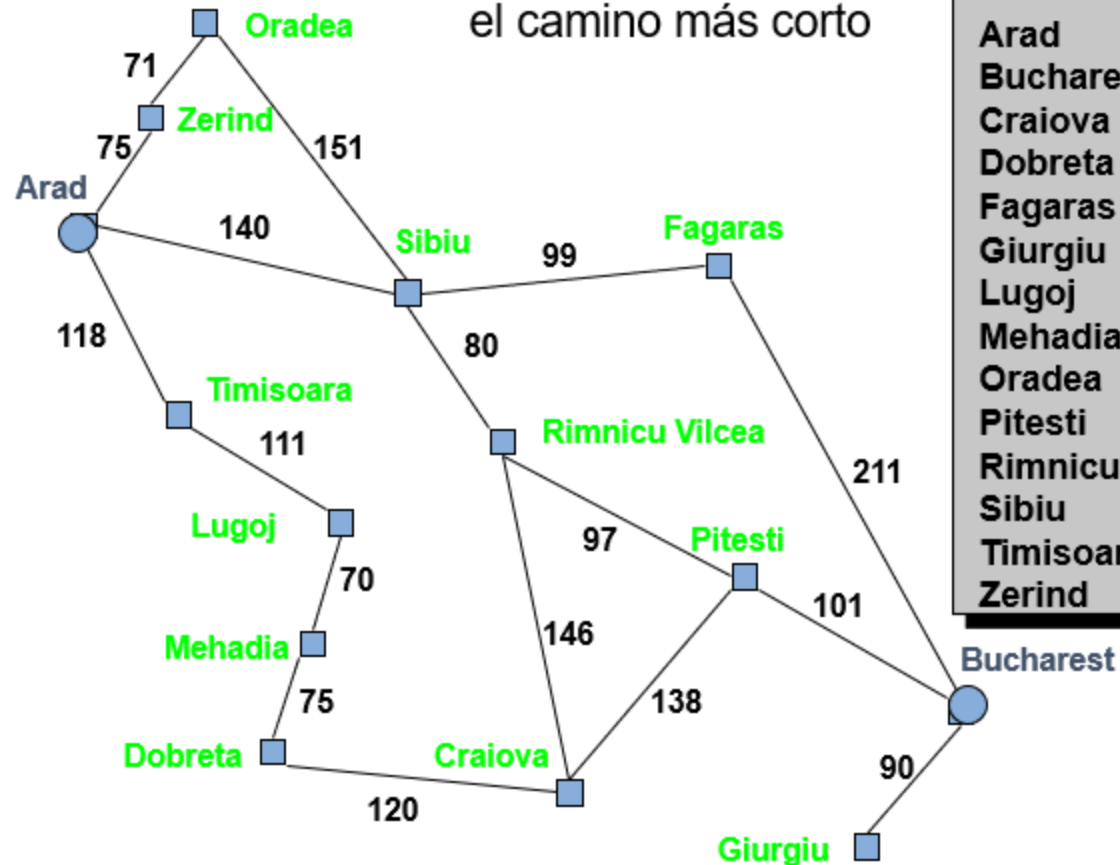
# A Estrategias Heurísticas: A Óptima (A\*)

Costos de recorridos en Rumania en Kms

Mapa de Rumania  
(Dist. entre Ciudades)

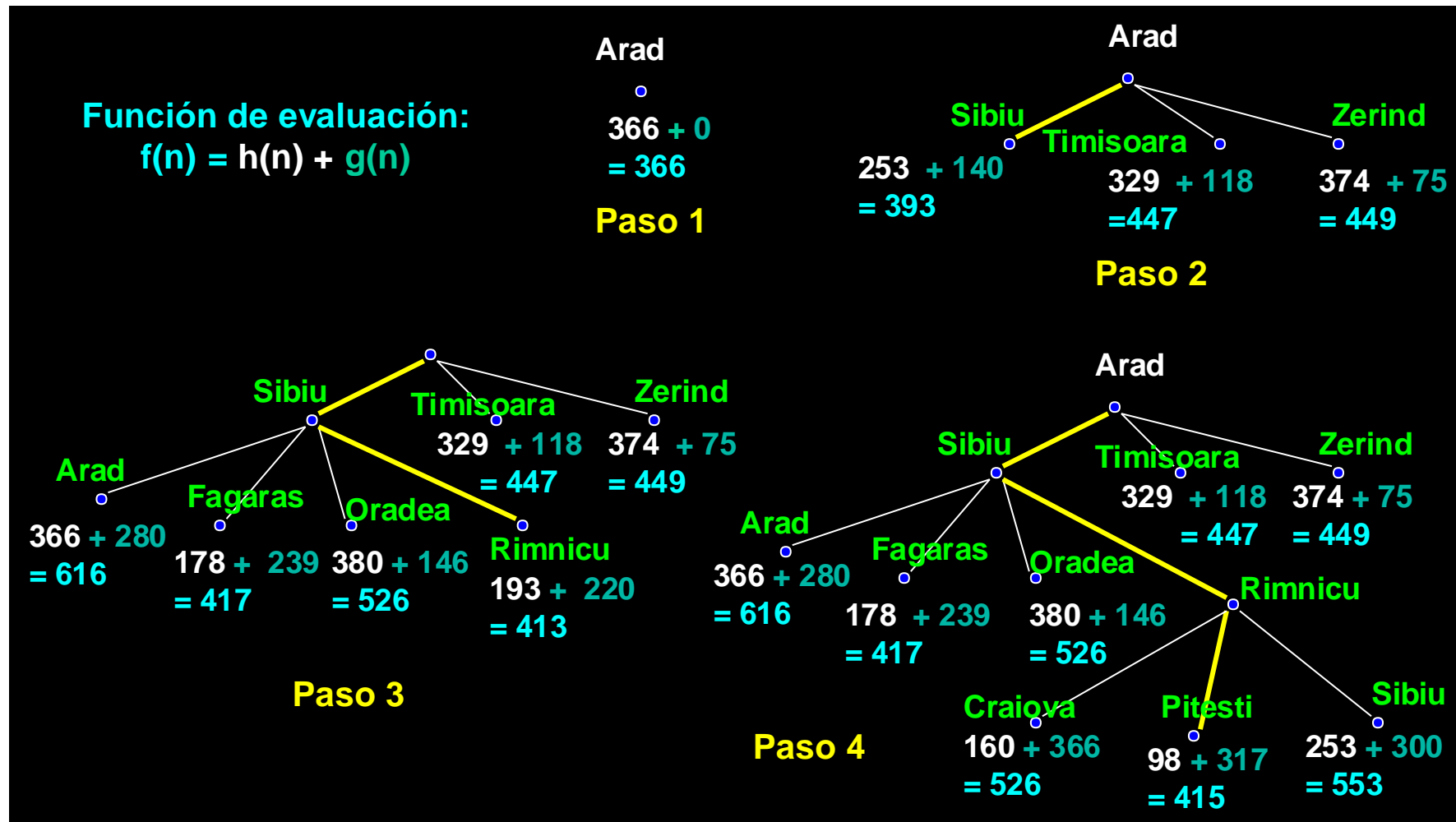
**Objetivo:** Ir de Arad a  
Bucharest utilizando  
el camino más corto

Dist. en línea recta  
a Bucharest (kms.)



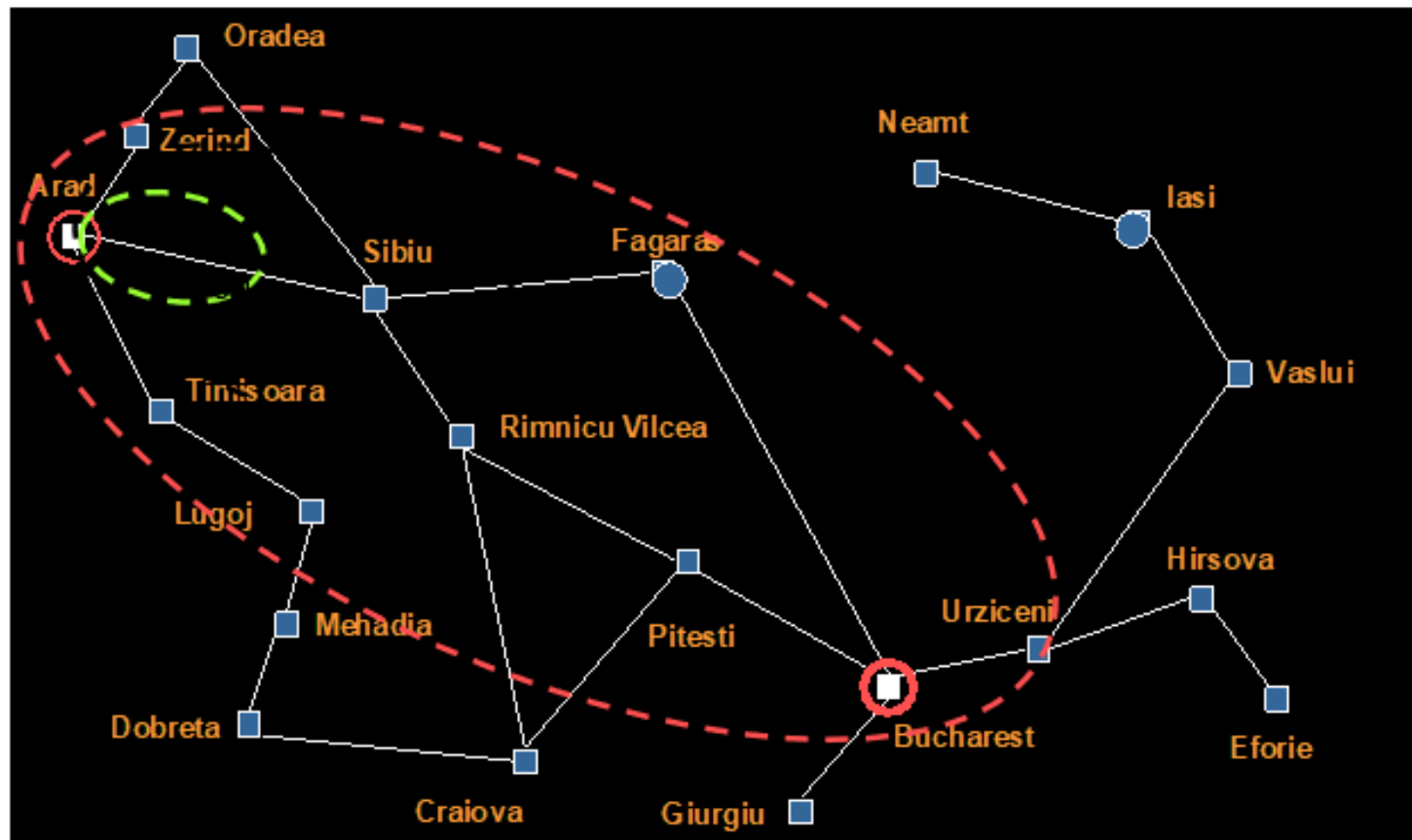
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Fagaras	178
Giurgiu	77
Lugoj	244
Mehadia	241
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Zerind	374

# A Estrategias Heurísticas: A Óptima (A\*)





# Estrategias Heurísticas: A Óptima (A\*)





# Evaluación de A Óptima ( $A^*$ )

- Completitud: Sí, en grafos localmente finitos
- Complejidad en tiempo: exponencial en error en  $h(n)$
- Complejidad en espacio: guarda todos los nodos en memoria
- Optimalidad: Sí

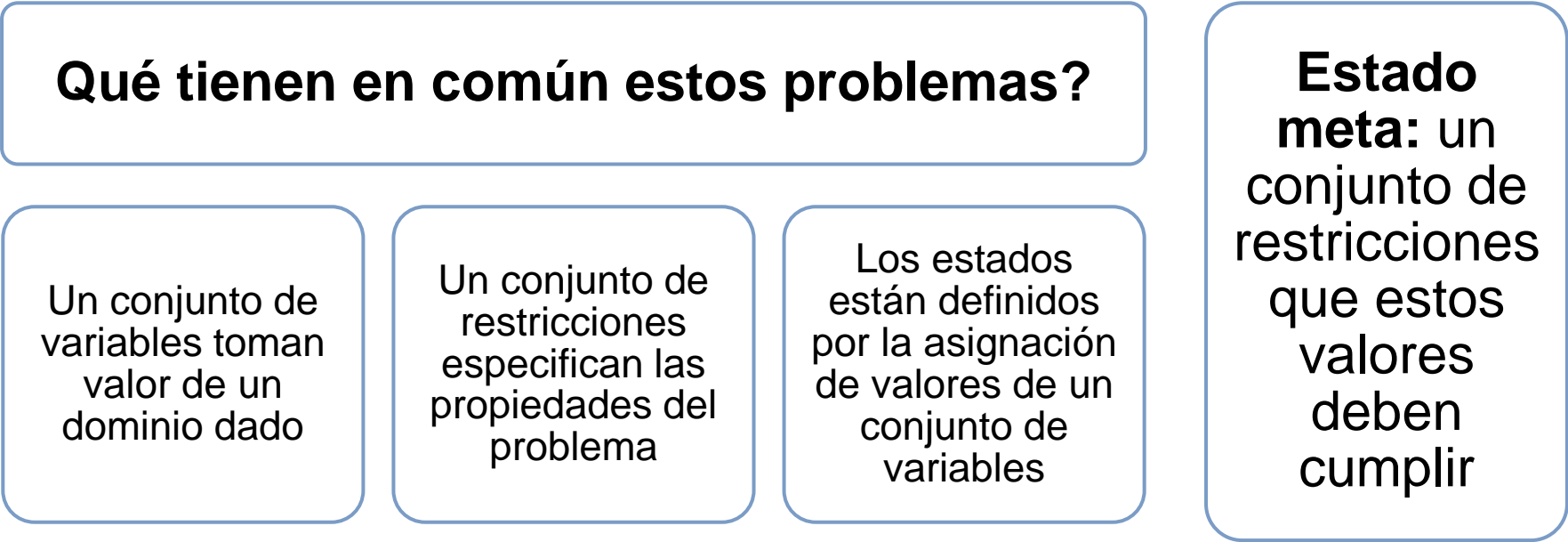


# Estrategias Heurísticas: Satisfacción de Restricciones

Cripto-aritmética: Asignar valores (0 - 9) a la letras para que la operación dada sea correcta

8 Reinas: Colocar 8 reinas en un tablero de ajedrez (8 x 8) de modo que ninguna esté en la misma fila, columna o diagonal que otra

Coloriz. de mapas: Colorear un mapa con **n** colores de forma tal que, países vecinos no estén de igual color



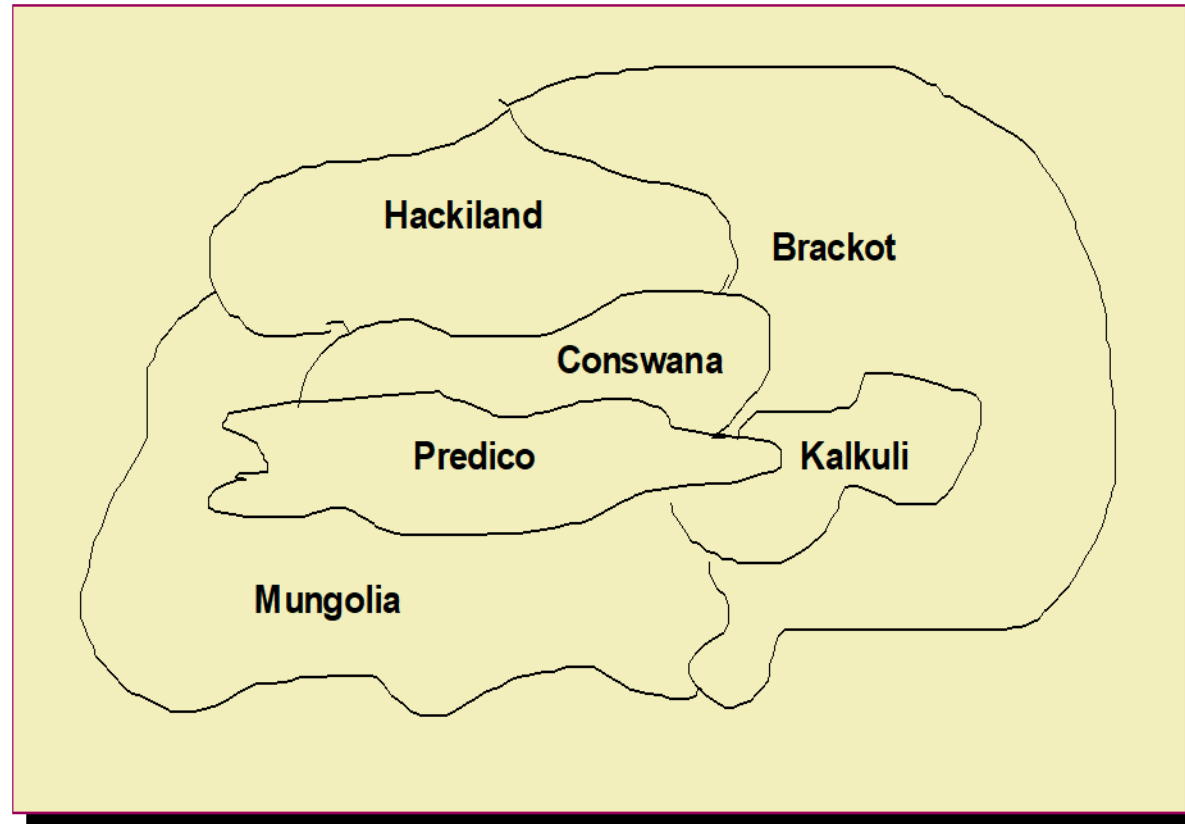


# Estrategias Heurísticas: Satisfacción de Restricciones

Qué heurísticas son buenas para este tipo de problemas?

**“Las que permiten elegir una variable para ser instanciada y escoger un valor para esa variable”**

Ej: Problema de Colorización de Mapa



Colores

Azul

Rojo

Rosado

Verde



# Estrategias Heurísticas: Satisfacción de Restricciones

**Variable más restringida:** en cada paso de la búsqueda, se asigna valor a la variable con menor número posibles valores.

**Variable más restrictiva:** asigna un valor a una variable que está involucrada en el mayor número de restricciones de otras variables aún no asignadas.

**Valor menos restrictivo:** escoge un valor que descarte el menor número de valores en las variables conectadas por restricciones a la variable actual.

El descenso de gradiente es un algoritmo de optimización que ajusta los parámetros (por ejemplo de una red neuronal), para minimizar la diferencia entre las predicciones y los valores reales. Se basa en conceptos de cálculo diferencial, específicamente en la derivada, que indica la dirección en la que el error disminuye más rápidamente. Se representa visualmente como moverse cuesta abajo en una gráfica de errores para encontrar el punto de menor pérdida.

En Búsqueda en el espacio de estados

- ❑ Considera todos los estados posibles a partir del estado actual y elige el mejor de ellos como nuevo estado.
- ❑ Dependiendo del objetivo de la búsqueda se denomina:
  - Descenso de gradiente: mínimo
  - Ascenso de colina: máximo

Funcionamiento del Algoritmo	Se calcula la pérdida usando una función de costo (ej. Mean Square Error - MSE).
	Se deriva la función de pérdida para encontrar la pendiente (gradiente).
	Se ajustan los parámetros de la red con la ecuación: $\text{coeficiente} = \text{coeficiente} - (\alpha \times \delta)$
donde::	$\alpha$ (alpha) es la tasa de aprendizaje (learning rate). $\delta$ (delta) es la dirección de ajuste del parámetro.
Elección de la tasa de aprendizaje:	Si es muy grande, la red oscila y no converge. Si es muy pequeña, el aprendizaje es muy lento.



# Búsqueda de Gradiente

## Problemas

---

**Puede quedar atrapado en:**

**Máximos locales:** generalmente aparecen en las cercanías de una solución .

---

**Mesetas:** áreas planas del espacio de búsqueda.

---

**Crestas:** tipo especial de máximos locales.

---

**Soluciones?**

Particulares y no seguras!!!

---



Gracias