

# Agentes para la resolución de problemas

Dr. Luis Felipe Marín Urias

## Agentes para la resolución de problemas

- Definición:

- Los agentes resolventes de problemas deciden qué hacer para encontrar secuencias de acciones que conduzcan a los estados deseables.
- Son “No Informados”, es decir, no dan información del problema salvo su definición.

## Agentes para la resolución de problemas

- Formulación del problema:
  - Estado Inicial
  - Acciones -> Función sucesor.
  - Espacio de Estados.
  - Secuencia de acciones (Camino).
  - Test Objetivo.
  - Costo del camino (Costo Individual).
  - Solución (Solución Óptima).

## Agentes para la resolución de problemas

- Forma restringida de un agente general:

```
function SIMPLE-PROBLEM-SOLVING-AGENT( p ) returns an action
  inputs: p, a percept
  static: s, an action sequence, initially empty
          state, some description of the current world state
          g, a goal, initially null
          problem, a problem formulation
  state ← UPDATE-STATE( state, p )
  if s is empty then
    g ← FORMULATE-GOAL( state )
    problem ← FORMULATE-PROBLEM( state, g )
    s ← SEARCH( problem )
  action ← RECOMMENDATION( s, state )
  s ← REMAINDER( s, state )
  return action
```

# Agentes para la resolución de problemas

- Ejemplo: Rumania

En unas vacaciones en Rumania; Actualmente se encuentra en la ciudad de Arad.

El vuelo de regreso sale mañana de Bucarest.

Formulación de Meta:

Ilegar a Bucarest

Formulación del problema:

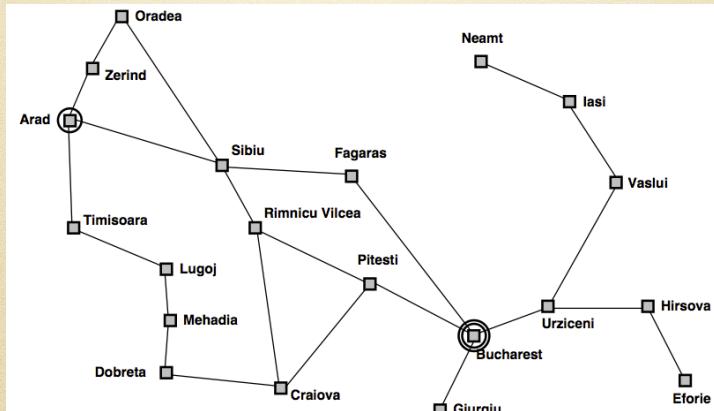
Estados: Muchas ciudades.

Operadores: Conducir entre las ciudades.

Encontrar la solución:

Secuencia de ciudades. Ejem. Arad, Sibiu, Fagaras, Bucarest.

## Mapa simplificado de Rumania



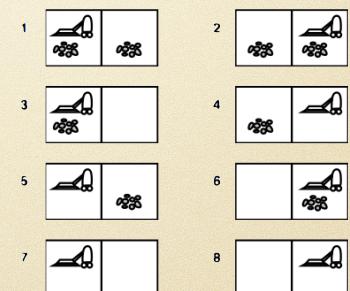
## Tipos de problemas

- Problemas de Juguete
- Problemas del mundo real

## Problemas de Juguete

- Estados:

- Aspiradora en la izquierda
- Aspiradora en la derecha
- Sucio derecho
- Sucio izquierdo.



$$2 \times 2^2 = 8$$

# Mundo de la aspiradora

- Estado inicial:

- cualquier estado



- Función Sucesor:

- Izquierda, derecha y Aspirar



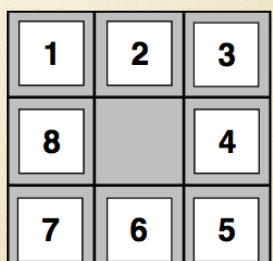
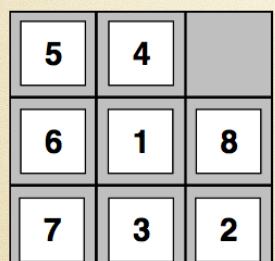
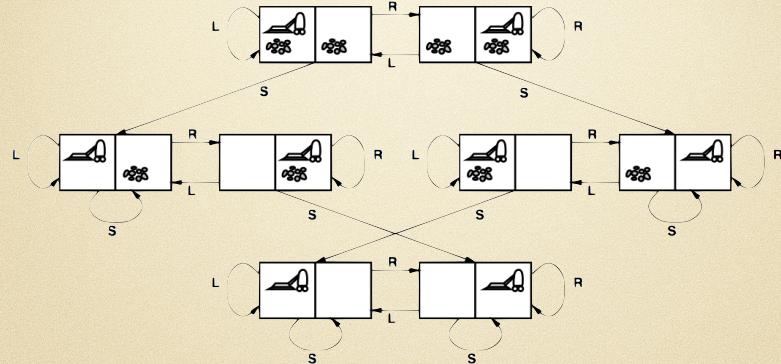
- Test objetivo:

- ¿Está limpio?

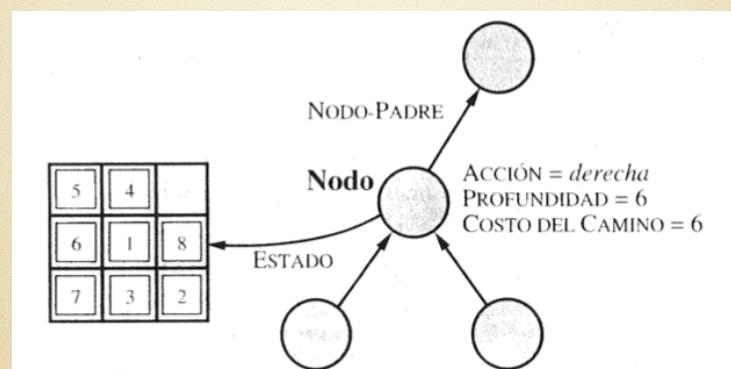


- Costo Camino:

- cada paso = 1



## Representación de los estados en nodos.

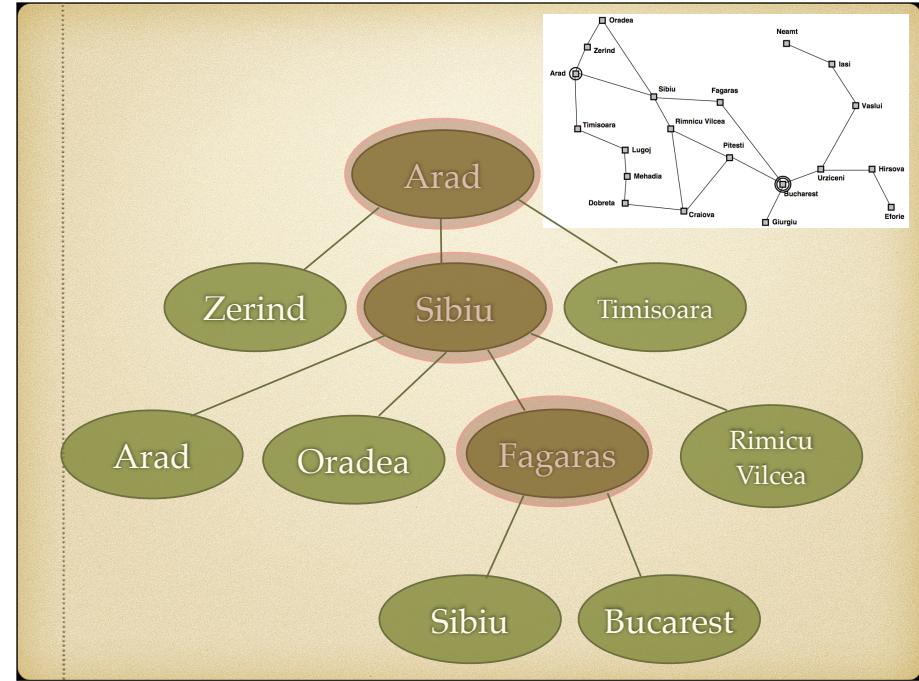


# Algoritmos de Búsqueda

- Idea Básica:

- Exploración simulada del espacio de estados; generando sucesores de los estados ya explorados. (Expandiendo Estados)

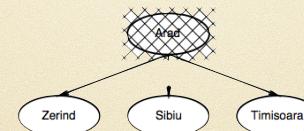
```
function GENERAL-SEARCH(problem, strategy) returns a solution, or failure
    initialize the search tree using the initial state of problem
    loop do
        if there are no candidates for expansion then return failure
        choose a leaf node for expansion according to strategy
        if the node contains a goal state then return the corresponding solution
        else expand the node and add the resulting nodes to the search tree
    end
```



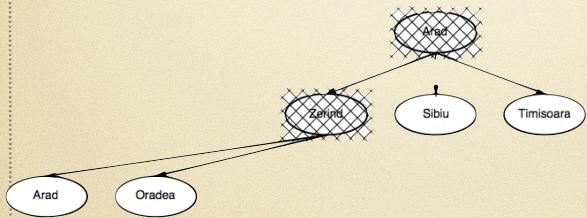
## Estrategias de búsqueda sin información

- Búsqueda Primero en Anchura
- Búsqueda Costo uniforme
- Búsqueda Primero en Profundidad
- Búsqueda de Profundización Iterativa.

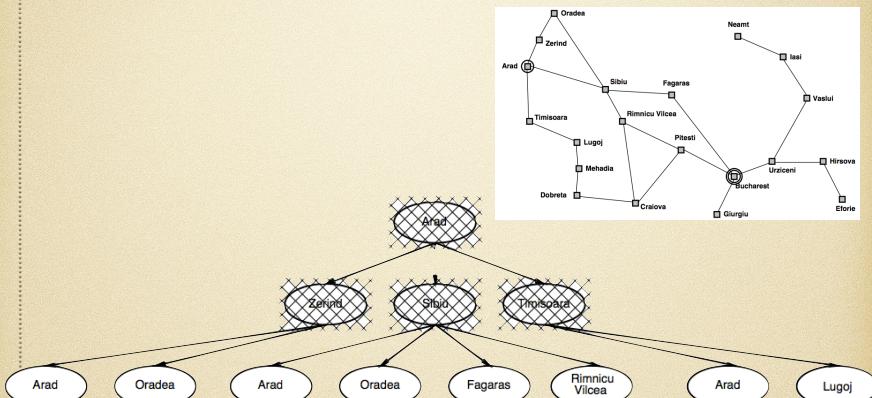
## Primero en Anchura



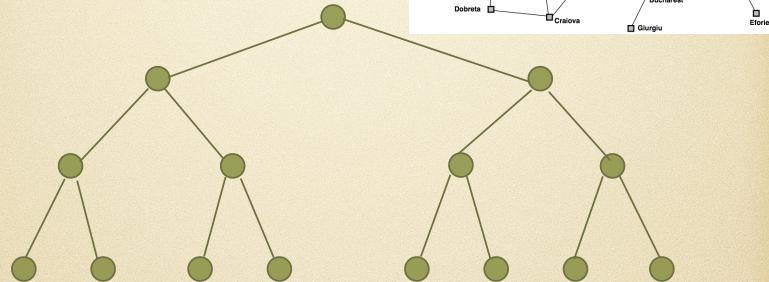
## Primero en Anchura



## Primero en Anchura



## Primero en Profundidad

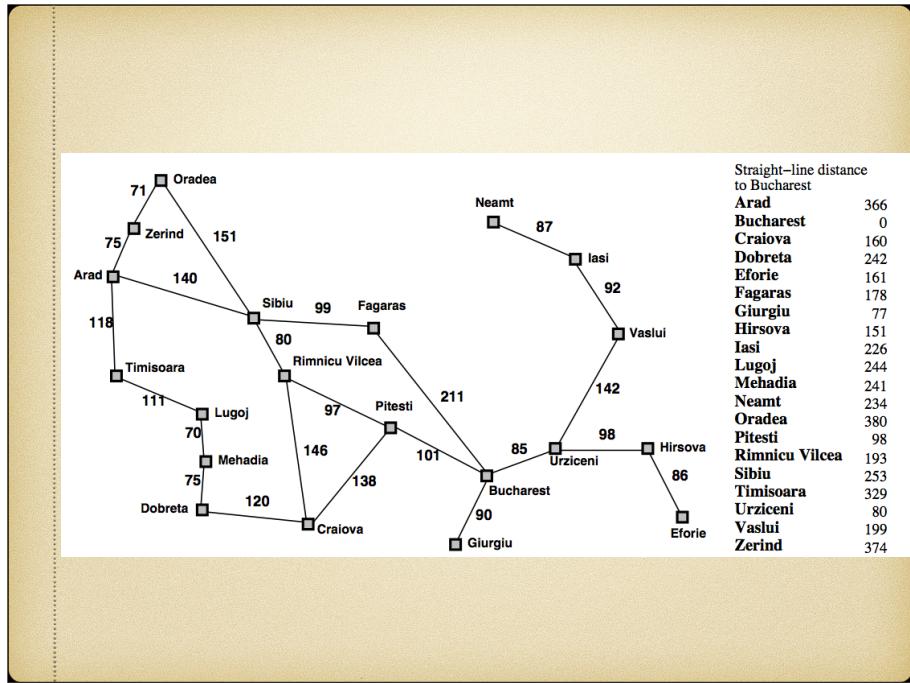


## Búsquedas informadas

- Primero el Mejor
- Búsqueda A\*
- Heurísticas

# Búsqueda de primero el mejor

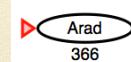
- Idea: Utilizar una función de evaluación en base a la información que se tenga. Estimar su nivel de "deseabilidad".
- Expandir el mas deseable nodo no expandido.
- franja: es un la cola ordenada en orden decreciente de deseabilidad.
- Casos:
  - Búsqueda Avara (voraz)
  - algoritmo A\*



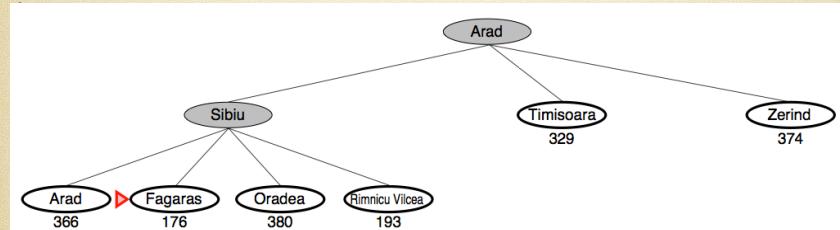
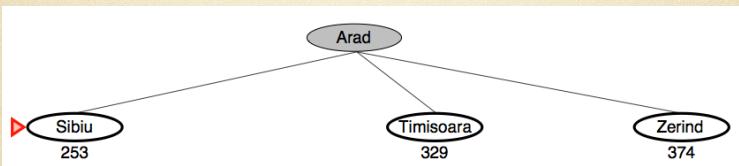
## Heurísticas

- Función de Evaluación  $h(n)$  - heurística de  $n$  = estimar el costo desde  $n$  hasta la meta mas cercana.
- Ejemplo :  $h(n) = \text{distancia en linea recta}$  desde  $n$  hasta Bucarest.
- El algoritmo greedy (avaro) expande el nodo que aparenta ser el más cercano al la meta

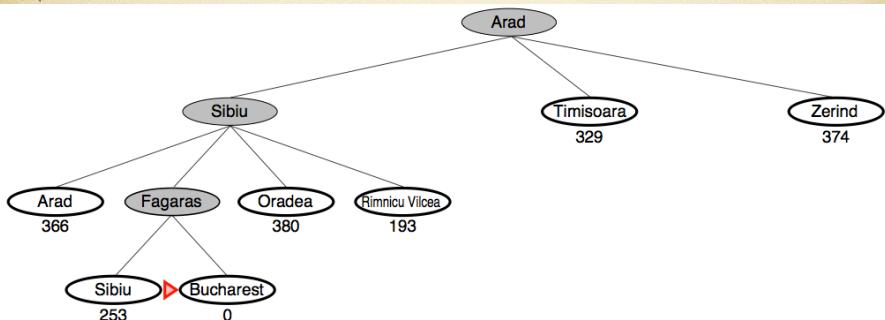
## Greedy



# Greedy



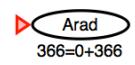
# Greedy



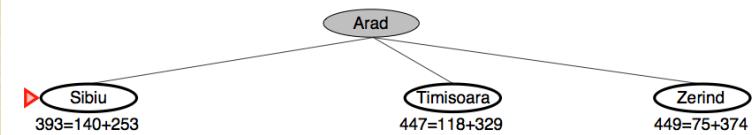
# A \*

- Idea: Evitar Rutas que ya son costosas
- Función de evaluación  $f(n) = g(n) + h(n)$ 
  - $g(n)$  = costo actual
  - $h(n)$  = costo estimado hasta la meta desde n.
  - $f(n)$  = Costo total estimado de la ruta desde n hasta la meta.

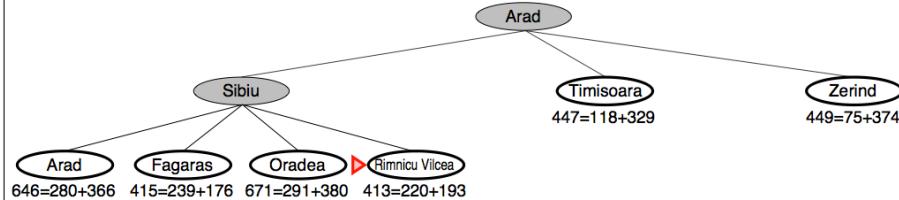
A \*



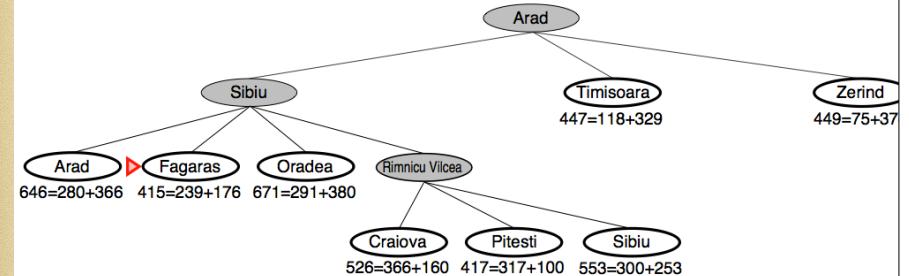
A \*



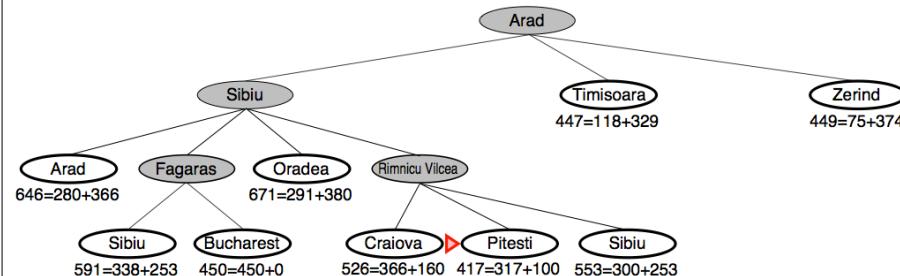
A \*



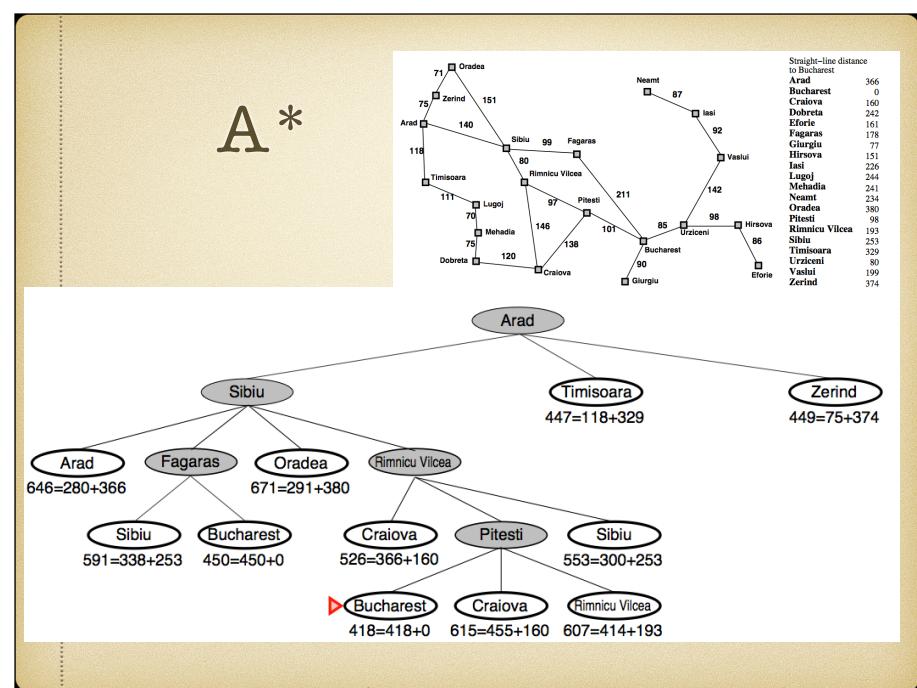
A \*



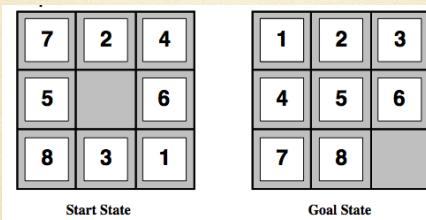
A \*



A \*



A \*



Heurísticas?

- $h_1(n)$ : Número de celdas mal colocadas
- $h_2(n)$ : Distancia total de Manhattan

## Heurísticas Admisibles

- $h(n)$  nunca debe sobreestimar el coste de alcanzar el objetivo.
- Las heurísticas son por naturaleza optimistas.
- $g(n) = \text{al coste real para llegar a } n$ .

# ¿Por qué no sobrestimar?

- Suponiendo en una búsqueda en árboles:
  - $G_2$  es un nodo en la frontera de un camino subóptimo.
  - $C^*$  es el costo de la solución óptima.
  - $h(G_2) = 0$ . Ciento para cualquier nodo objetivo.
  - $f(n) = g(n) + h(n) = g(n) > C^*$

# ¿Por qué no sobrestimar?

- Si  $n$  es un nodo en la frontera perteneciente al camino óptimo.
- $f(n) = g(n) + h(n) \leq C^*$

d	Costo de la búsqueda			Factor de ramificación eficaz		
	BPI	A*( $h_1$ )	A*( $h_2$ )	BPI	A*( $h_1$ )	A*( $h_2$ )
2	10	6	6	2,45	1,79	1,79
4	112	13	12	2,87	1,48	1,45
6	680	20	18	2,73	1,34	1,30
8	6384	39	25	2,80	1,33	1,24
10	47127	93	39	2,79	1,38	1,22
12	3644035	227	73	2,78	1,42	1,24
14	—	539	113	—	1,44	1,23
16	—	1301	211	—	1,45	1,25
18	—	3056	363	—	1,46	1,26
20	—	7276	676	—	1,47	1,27
22	—	18094	1219	—	1,48	1,28
24	—	39135	1641	—	1,48	1,26

Figura 4.8 Comparación de los costos de la búsqueda y factores de ramificación eficaces para la BÚSQUEDA-PROFUNDIDAD-ITERATIVA y los algoritmos A\* con  $h_1$  y  $h_2$ . Los datos son la media de 100 ejemplos del puzzle-8, para soluciones de varias longitudes.