

# Artificial Intelligence

## Lecture03 – Heuristic search & Optimality



# Contenido

1. Algoritmos de Búsqueda informada
2. Greedy Search
3. A\* Search
4. Optimalidad
5. Recursive Best First Search
6. Ejercicio Maze-Algorithm



# Repaso

## Search problem:

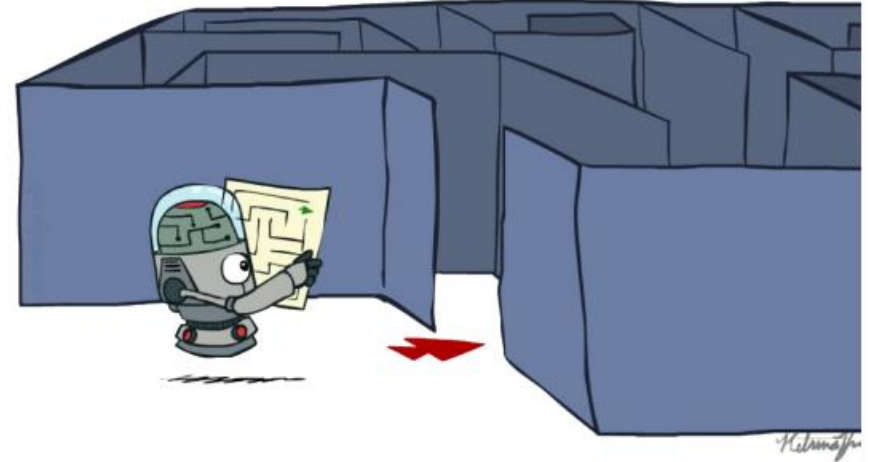
- States (configurations of the world)
- Actions and costs
- Successor function (world dynamics)
- Start state and goal test

## Search tree:

- Nodes: represent plans for reaching states
- Plans have costs (sum of action costs)

## Search algorithm:

- Systematically builds a search tree
- Chooses an ordering of the fringe (unexplored nodes)
- Optimal: finds least-cost plans



# Agentes de Búsqueda Informada

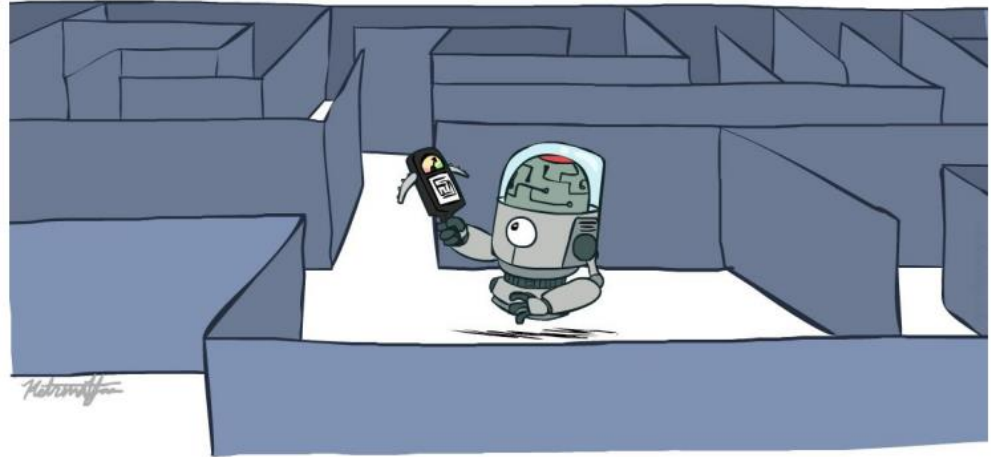
The basic lesson of this lecture: it is **MUCH** easier to search when you have some idea where you're going!



# Agentes de Búsqueda Informada

Las técnicas informadas utilizan pistas específicas del dominio sobre la ubicación de los objetivos

Las pistas vienen en forma de una función heurística, denotada como  $h(n)$ .

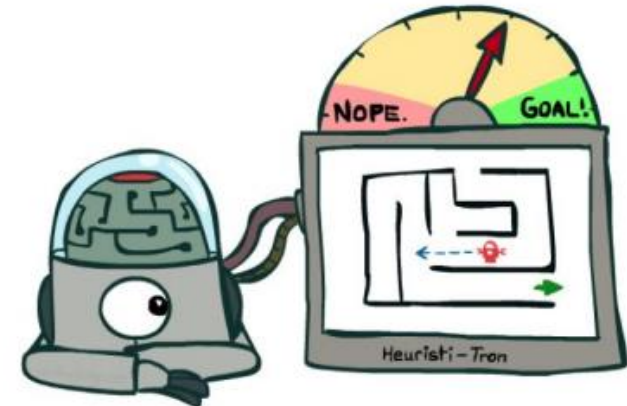
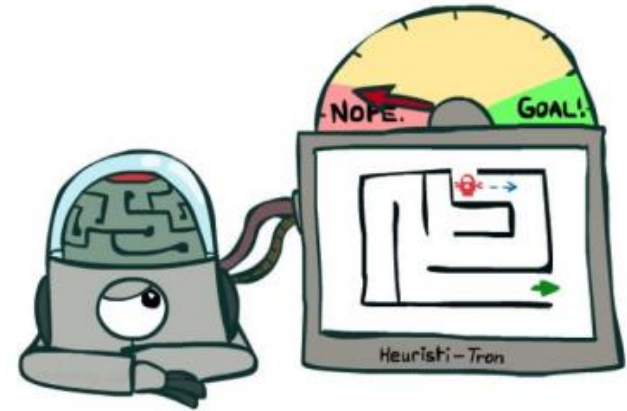


# Agentes de Búsqueda Informada

Y entonces... ¿Que es una Heurística?

*“This additional information, helps estimate the cost or distance from a given node ... to the goal node.”*

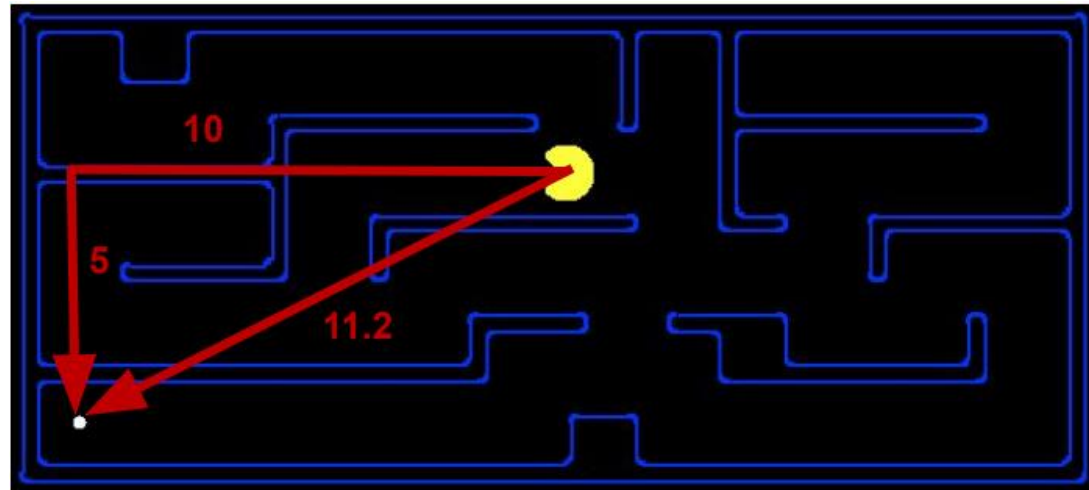
*“Heuristics are mental shortcuts or rules of thumb that simplify decision making and problem-solving processes. ... They rely on intuitive judgment and common sense rather than a systematic and exhaustive analysis of every possible option.”*



# Agentes de Búsqueda Informada

A heuristic is:

- A function that *estimates* how close a state is to a goal
- Designed for a particular search problem
- Examples: Manhattan distance, Euclidean distance for pathing



## Tipos de Heurísticas

Tipo de Heurística	Definición	Ejemplo práctico (profesional/empresarial)
Disponibilidad	Se basa en la facilidad con la que recordamos ejemplos o información reciente. Se asume que lo más “fácil de recordar” es lo más probable.	Un gerente de riesgos sobreestima la probabilidad de fallas porque recientemente ocurrió un problema visible en otro proyecto.
Representatividad	Se juzga la probabilidad de un evento según qué tan similar parece a un caso típico, ignorando datos estadísticos reales.	Un reclutador asume que un candidato que viste de forma “corporativa” será necesariamente el más competente, sin revisar sus métricas reales de desempeño.
Anclaje y ajuste	Se toma un valor inicial (ancla) y se hacen ajustes a partir de él.	En una negociación de salarios, el primer valor mencionado influye fuertemente en el acuerdo final, incluso si no refleja el valor real del puesto.
Reconocimiento	Se selecciona la opción que se reconoce más fácilmente, bajo la suposición de que lo conocido es mejor o más seguro.	Un consumidor elige una marca de café porque “le suena” más confiable que otras desconocidas, aunque no compare precios ni calidad.
Regla de parada (stopping rule)	Se detiene la búsqueda de información cuando se encuentra una opción “suficientemente buena”, sin agotar las alternativas.	Un equipo de desarrollo selecciona la primera herramienta de software que cumple los requisitos básicos sin evaluar opciones potencialmente más eficientes.





# Gredy Search



# Greedy Search

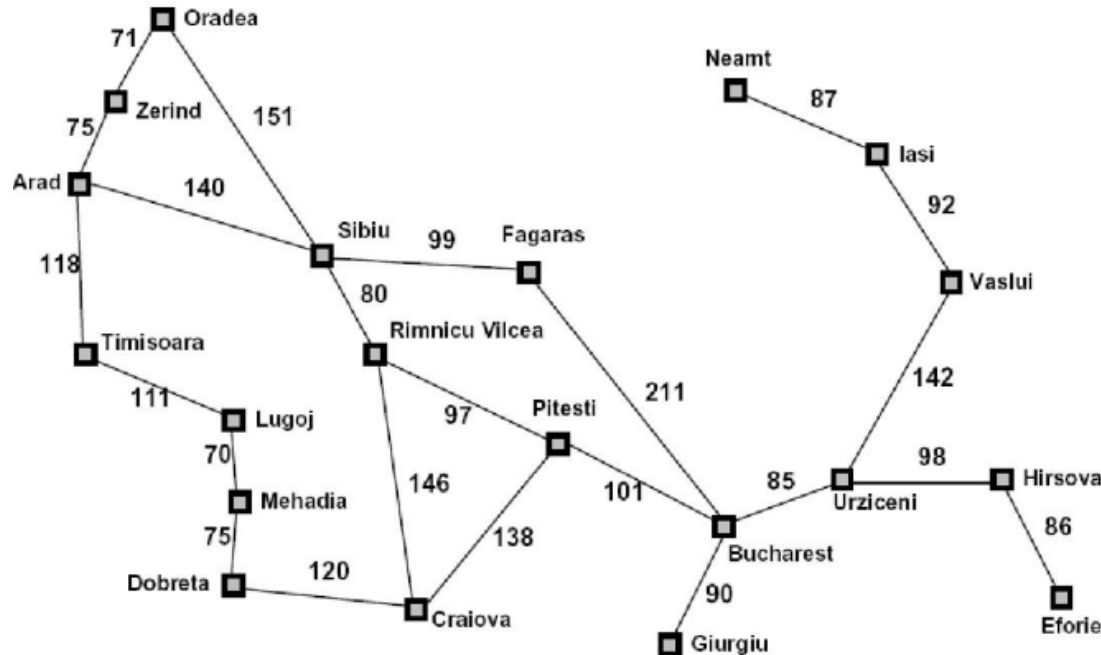
Es una forma de aplicación de la búsqueda Best-First Search que expande primero el nodo con el valor  $h(n)$  más bajo (el nodo que parece estar más cerca del objetivo), con el argumento de que esto probablemente llevará a una solución rápidamente.

$$f(n) = h(n).$$



# Greedy Best-First

Usaremos la heurística de distancia en línea recta, que llamaremos  $h_{SLD}$



Straight-line distance  
to Bucharest

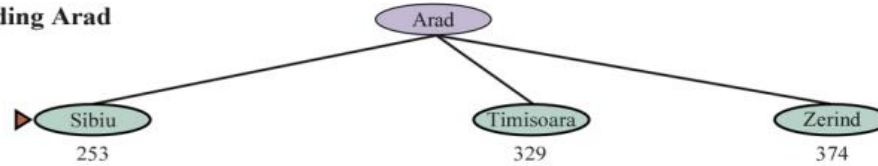
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

# Greedy Search

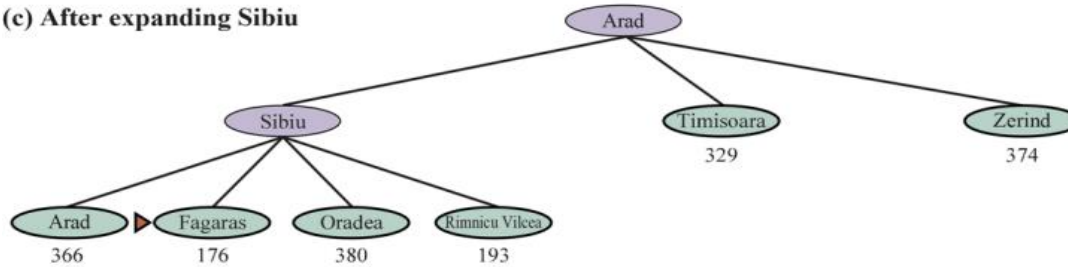
(a) The initial state



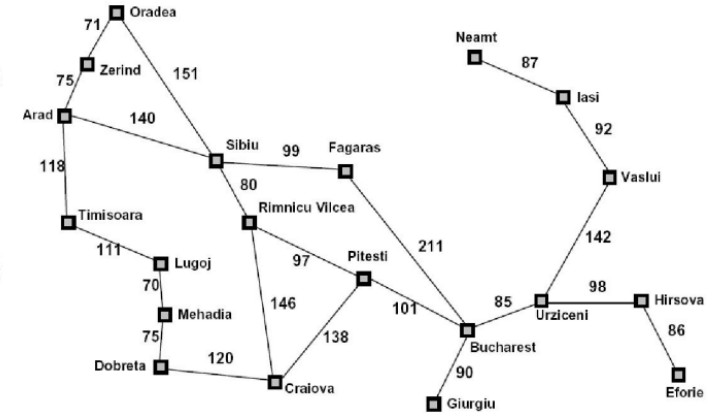
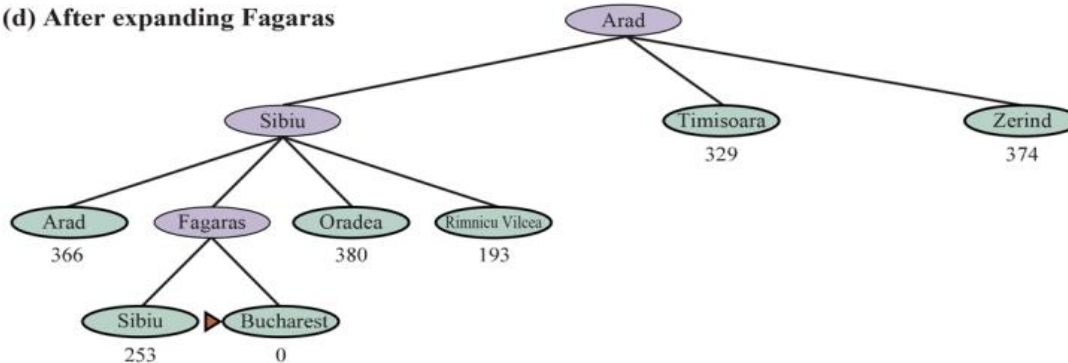
(b) After expanding Arad



(c) After expanding Sibiu



(d) After expanding Fagaras



What can go wrong?



# Greedy Search

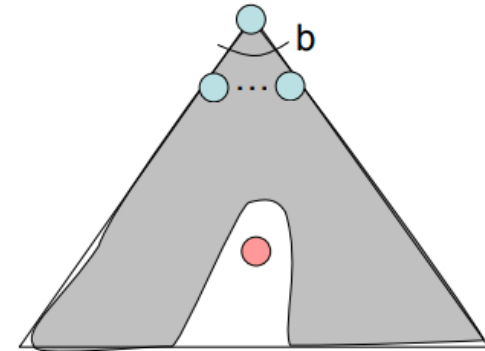
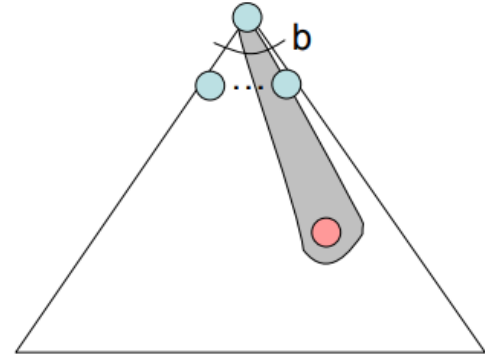
Strategy: expand a node that you think is closest to a goal state

- Heuristic: estimate of distance to nearest goal for each state

A common case:

- Best-first takes you straight to the (wrong) goal

Worst-case: like a badly-guided DFS



# A\* Search



# A\* Search

El algoritmo de búsqueda informada más común es la búsqueda A\* search (A-star search), una búsqueda best-first que utiliza la función de evaluación:

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

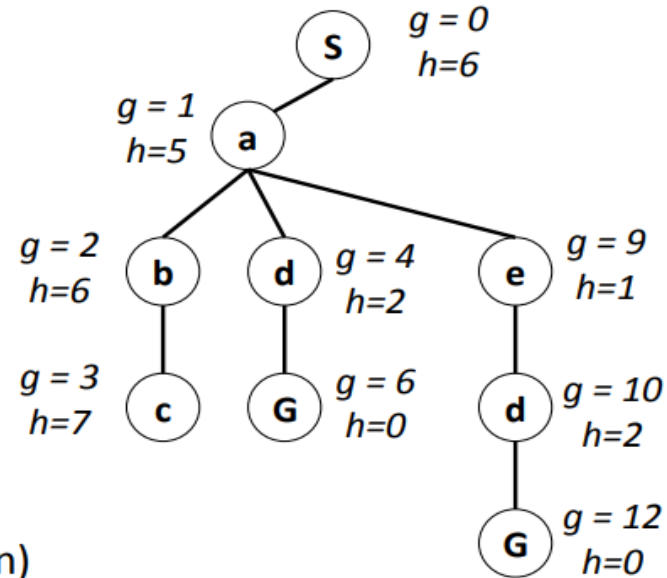
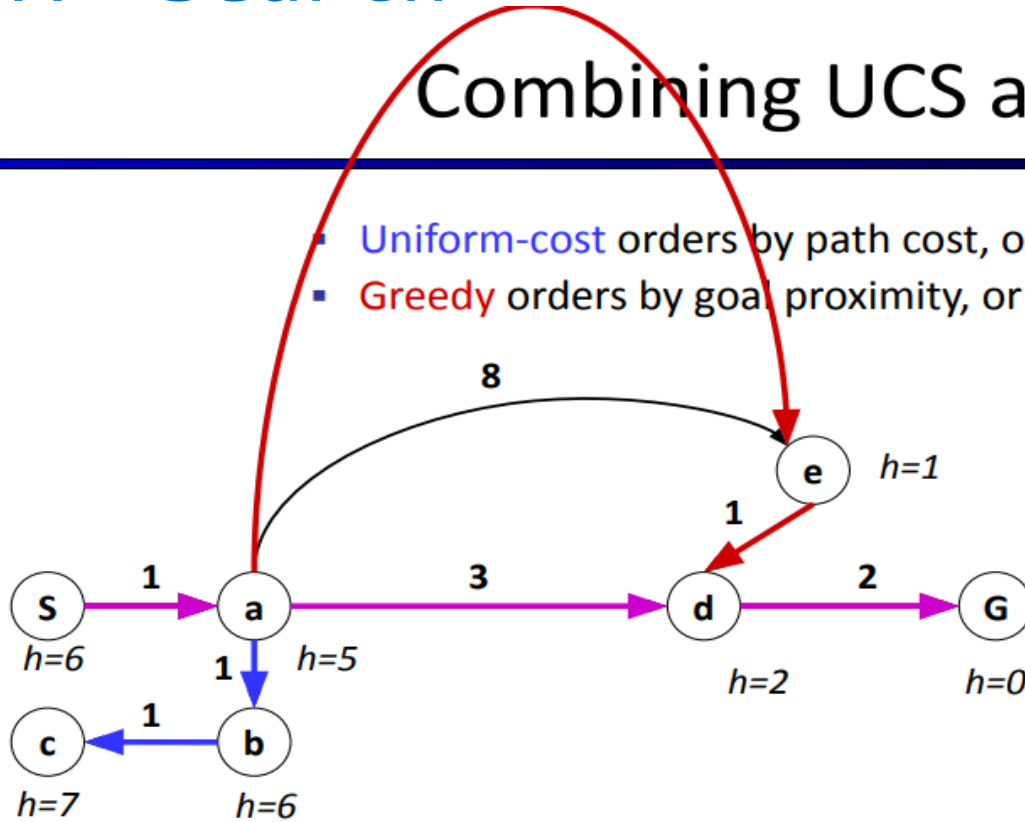
Donde  $g(n)$  es el costo del camino desde el estado inicial hasta el nodo  $n$ , y  $h(n)$  es el costo estimado del camino más corto desde  $n$  hasta un estado objetivo



# A\* Search

## Combining UCS and Greedy

- Uniform-cost orders by path cost, or *backward cost*  $g(n)$
- Greedy orders by goal proximity, or *forward cost*  $h(n)$



- A\* Search orders by the sum:  $f(n) = g(n) + h(n)$

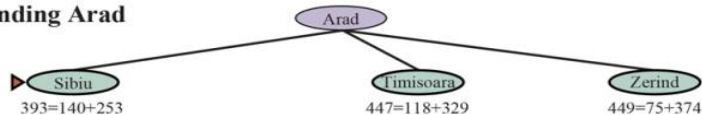


# A\* Search

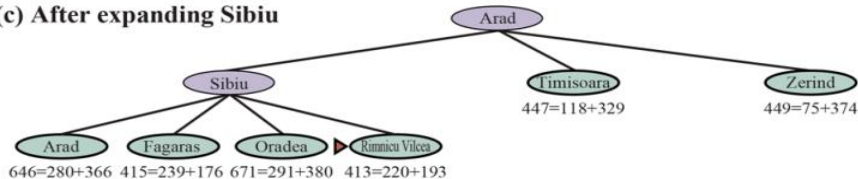
(a) The initial state



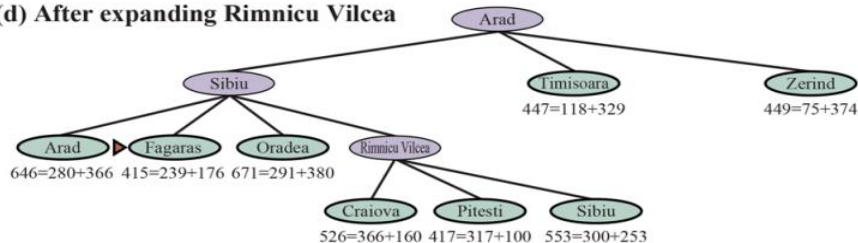
(b) After expanding Arad



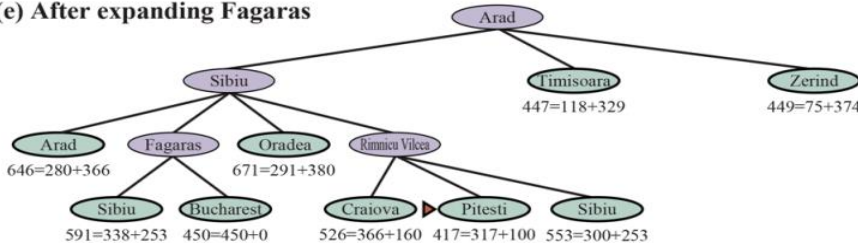
(c) After expanding Sibiu



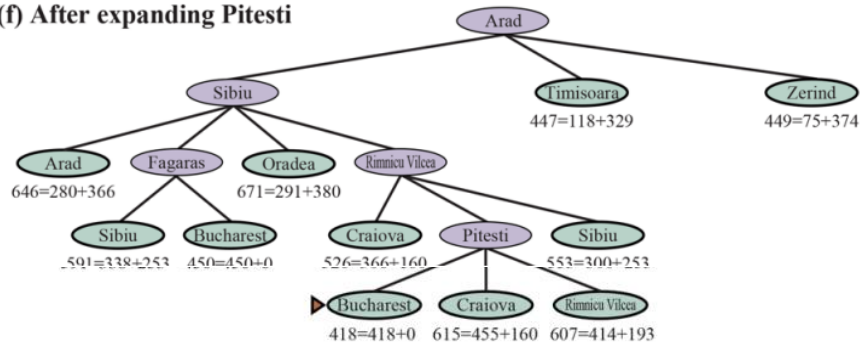
(d) After expanding Rimnicu Vilcea



(e) After expanding Fagaras

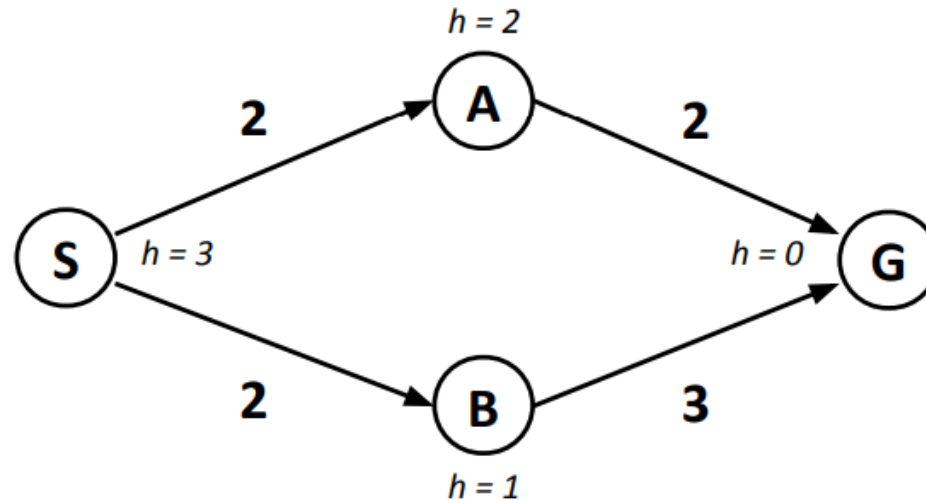


(f) After expanding Pitesti



# A\* Search

Should we stop when we enqueue a goal (i.e., add to fringe)?



Order in which we  
add to fringe:

S

S→A

S→B

Expand B first

S→B→G (cost 5)

Don't declare victory  
yet! Be patient!

S→A→G (cost 4)

Victory!

No: only stop when we dequeue a goal



# A\* Search

## CARACTERISTICAS:

La búsqueda A\* es **completa**.

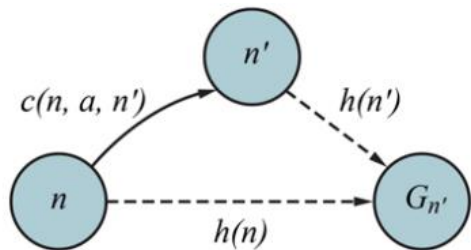
Que sea **optima** en costo depende de:

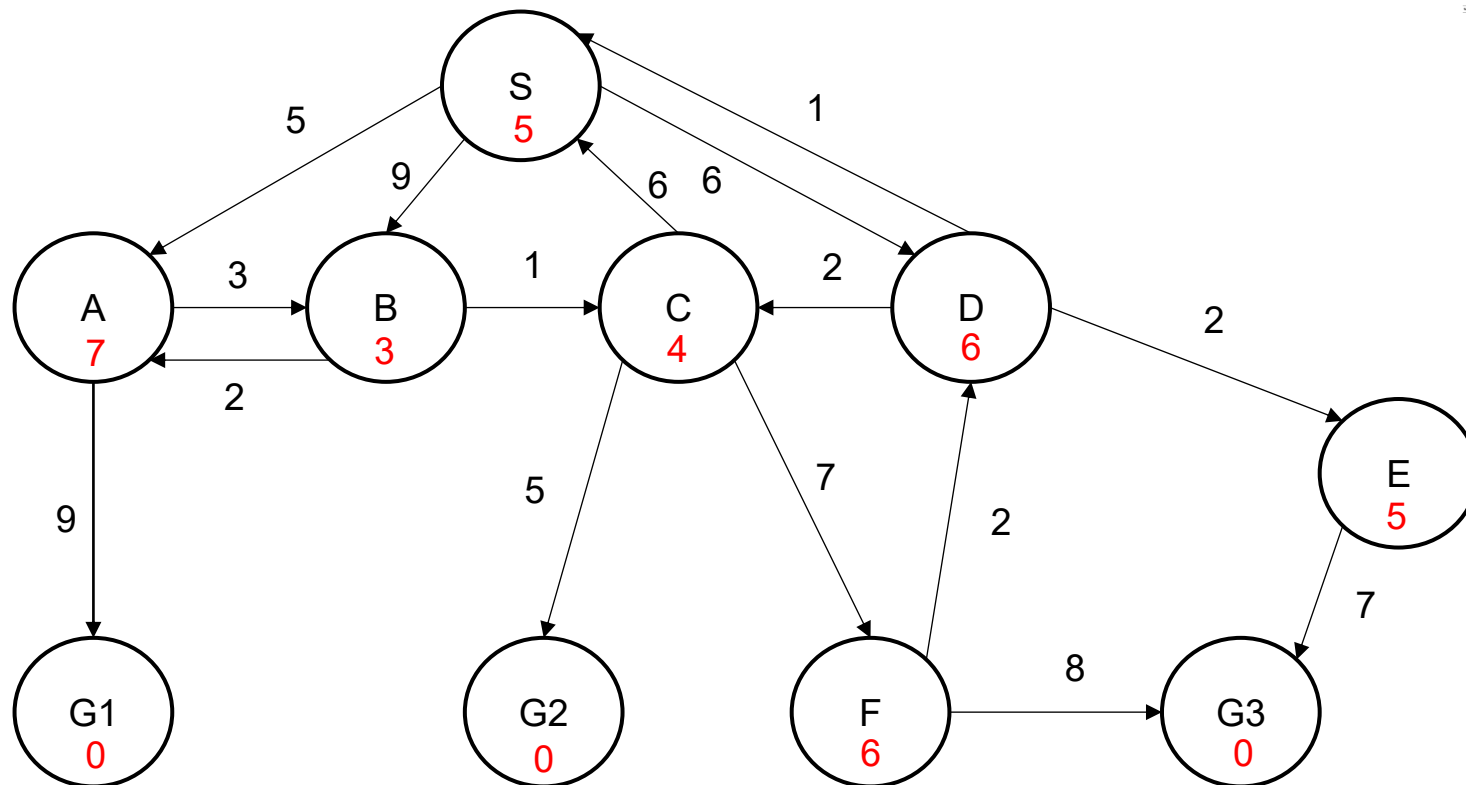
**Admisibilidad:** Dado que se cumpla que  $h(n) \leq h^*(n)$ .

Donde  $h^*(n)$  es el costo real mínimo desde n hasta el objetivo.

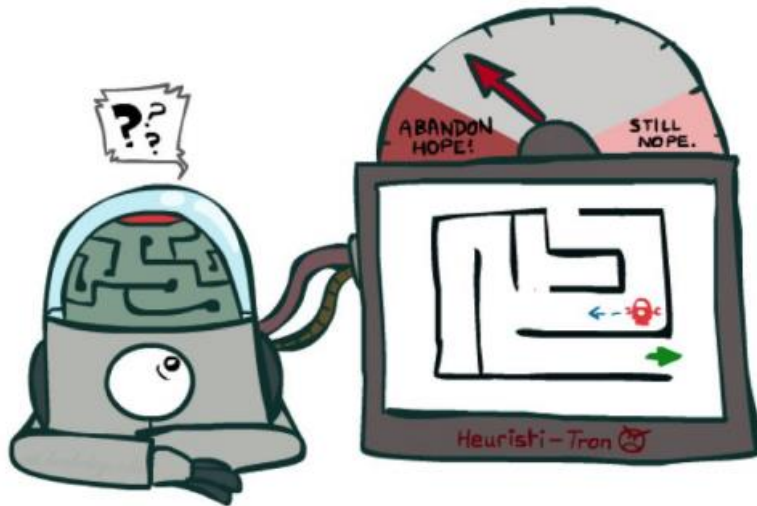
**Consistencia:** Una heurística  $h(n)$  es consistente si, para cada nodo n y cada sucesor  $n'$  de n generado por una acción a, tenemos:

$$h(n) \leq c(n, a, n') + h(n').$$

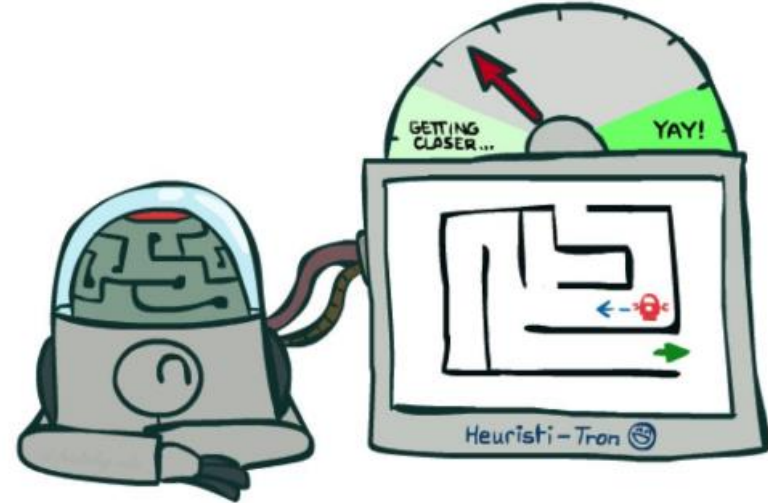




## Idea: Admissibility



Inadmissible (pessimistic) heuristics break optimality by trapping good plans on the fringe



Admissible (optimistic) heuristics slow down bad plans but never outweigh true costs



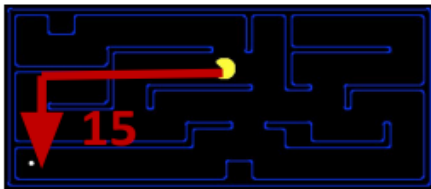
## Admissible Heuristics

A heuristic  $h$  is *admissible* (optimistic) if:

$$0 \leq h(n) \leq h^*(n)$$

where  $h^*(n)$  is the true cost to a nearest goal

Examples:



4



Coming up with admissible heuristics is most of what's involved in using A\* in practice.



## CONCLUSIÓN:

A\* Search puede ser completa, óptima en costo y óptimamente eficiente

El inconveniente es que, para muchos problemas, el número de nodos expandidos puede ser exponencial en relación con la longitud de la solución.



# Optimalidad

“An algorithm is complete if it is guaranteed to find a solution if one exists, and it is optimal if it always finds the best solution.”

“Optimality is the property that the algorithm finds the best trajectory, provided one exists.”





# Optimalidad

La búsqueda A\* tiene muchas cualidades positivas, pero expande muchos nodos. Podemos explorar menos nodos (utilizando menos tiempo y espacio) si estamos dispuestos a aceptar soluciones que no sean óptimas, pero que sean "suficientemente buenas": lo que llamamos **soluciones satisfactorias**.

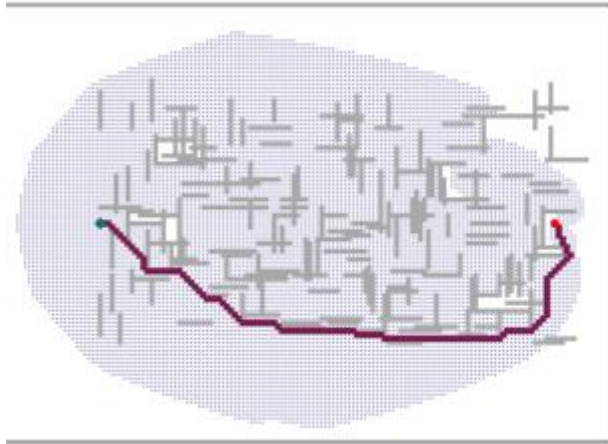
Si permitimos que la búsqueda A\* utilice una heurística inadmisibles (una que puede sobreestimar), podemos reducir la cantidad de nodos expandidos

$$f(n) = g(n) + W \cdot h(n) \quad \text{para algún } W > 1$$

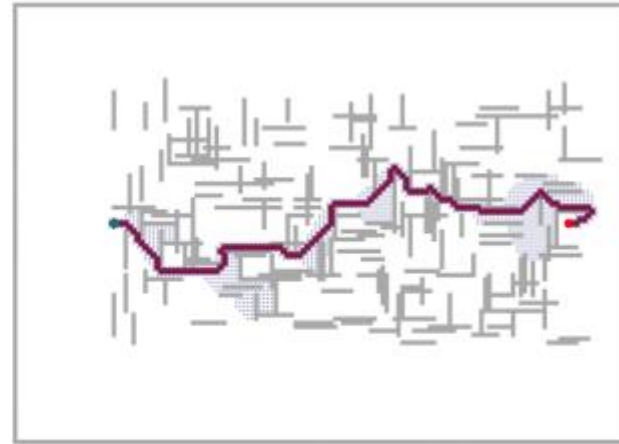


# Optimalidad

En general, si la solución óptima cuesta  $C^*$ , una *Weighted A\* Search* encontrará una solución que cuesta entre  $C^*$  y  $w \times C^*$  ; pero en la práctica, usualmente obtenemos resultados mucho más cercanos a  $C^*$  que a  $w \times C^*$



(a)



(b)



# Optimalidad

A\* search:  $g(n) + h(n)$  ( $W = 1$ )

Uniform-cost search:  $g(n)$  ( $W = 0$ )

Greedy best-first search:  $h(n)$  ( $W = \infty$ )

Weighted A\* search:  $g(n) + W \times h(n)$  ( $1 < W < \infty$ )



Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes <sup>1</sup>	Yes <sup>1,2</sup>	No	No	Yes <sup>1</sup>	Yes <sup>1,4</sup>
Optimal cost?	Yes <sup>3</sup>	Yes	No	No	Yes <sup>3</sup>	Yes <sup>3,4</sup>
Time	$O(b^d)$	$O(b^{1+\lceil C^*/\epsilon \rceil})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+\lceil C^*/\epsilon \rceil})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$



# Recursive best-first search

La memoria se divide entre la frontera y los estados alcanzados. En nuestra implementación de búsqueda mejor primero, un estado que está en la frontera se almacena en dos lugares: como un nodo en la frontera (para que podamos decidir qué expandir a continuación) y como una entrada en la tabla de estados alcanzados (para saber si hemos visitado el estado antes).

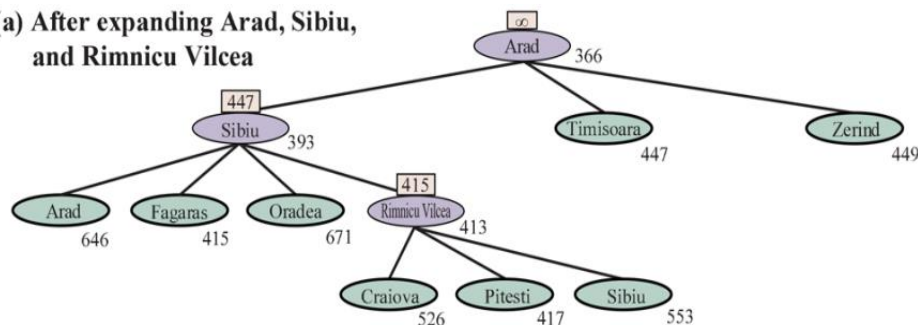
Recursive Best-First Search (RBFS) es uno de los algoritmos diseñados para conservar el uso de la memoria



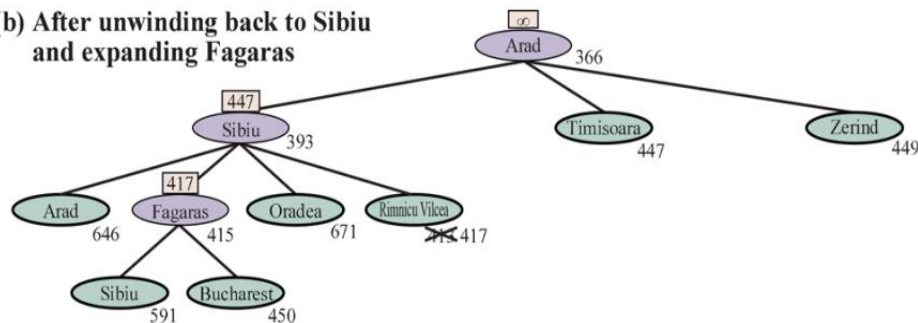
# Recursive best-first search

RBFS se asemeja a una búsqueda recursiva en profundidad, pero en lugar de continuar indefinidamente por el camino actual, utiliza la variable *limit* para hacer un seguimiento del valor  $f$  de la mejor ruta alternativa disponible desde cualquier antecesor del nodo actual

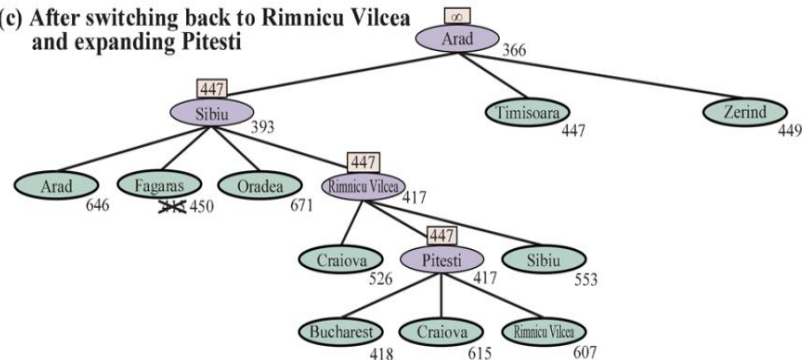
(a) After expanding Arad, Sibiu, and Rimnicu Vilcea



(b) After unwinding back to Sibiu and expanding Fagaras



(c) After switching back to Rimnicu Vilcea and expanding Pitesti



# Recursive best-first search

**function** RECURSIVE-BEST-FIRST-SEARCH(*problem*) **returns** a solution or *failure*  
    *solution, fvalue*  $\leftarrow$  RBFS(*problem*, NODE(*problem*.INITIAL),  $\infty$ )  
**return** *solution*

**function** RBFS(*problem, node, f\_limit*) **returns** a solution or *failure*, and a new *f*-cost limit  
    **if** *problem*.IS-GOAL(*node*.STATE) **then return** *node*  
    *successors*  $\leftarrow$  LIST(EXPAND(*node*))  
    **if** *successors* is empty **then return** *failure,  $\infty$*   
    **for each** *s* **in** *successors* **do**       // update *f* with value from previous search  
        *s.f*  $\leftarrow$  max(*s*.PATH-COST + *h*(*s*), *node.f*)  
    **while true do**  
        *best*  $\leftarrow$  the node in *successors* with lowest *f*-value  
        **if** *best.f* > *f\_limit* **then return** *failure, best.f*  
        *alternative*  $\leftarrow$  the second-lowest *f*-value among *successors*  
        *result, best.f*  $\leftarrow$  RBFS(*problem, best, min(f\_limit, alternative)*)  
        **if** *result*  $\neq$  *failure* **then return** *result, best.f*



# Recursive best-first search

CODE AND CLASS EXERCISE 1





# Recursive best-first search

## GENERAL CLASS EXERCISE 2 MAZE ALGORITHM



# References

GeeksforGeeks. (2025, abril 7). *Informed Search Algorithms in Artificial Intelligence*. Recuperado de <https://www.geeksforgeeks.org/artificial-intelligence/informed-search-algorithms-in-artificial-intelligence/>

DeepAI. (s.f.). *Heuristics*. En *DeepAI machine Learning glossary*. Recuperado de <https://deepai.org/machine-learning-glossary-and-terms/heuristics>

Berkeley University (CS188). (2022). *Completeness and Optimality - A Search\**. En *CS188 Lecture Notes*. Recuperado de <https://inst.eecs.berkeley.edu/~cs188/fa22/assets/notes/cs188-fa22-note02.pdf>

Russell, S. J., & Norvig, P. (2020). *Artificial intelligence: A modern approach* (4th ed.). Pearson.

