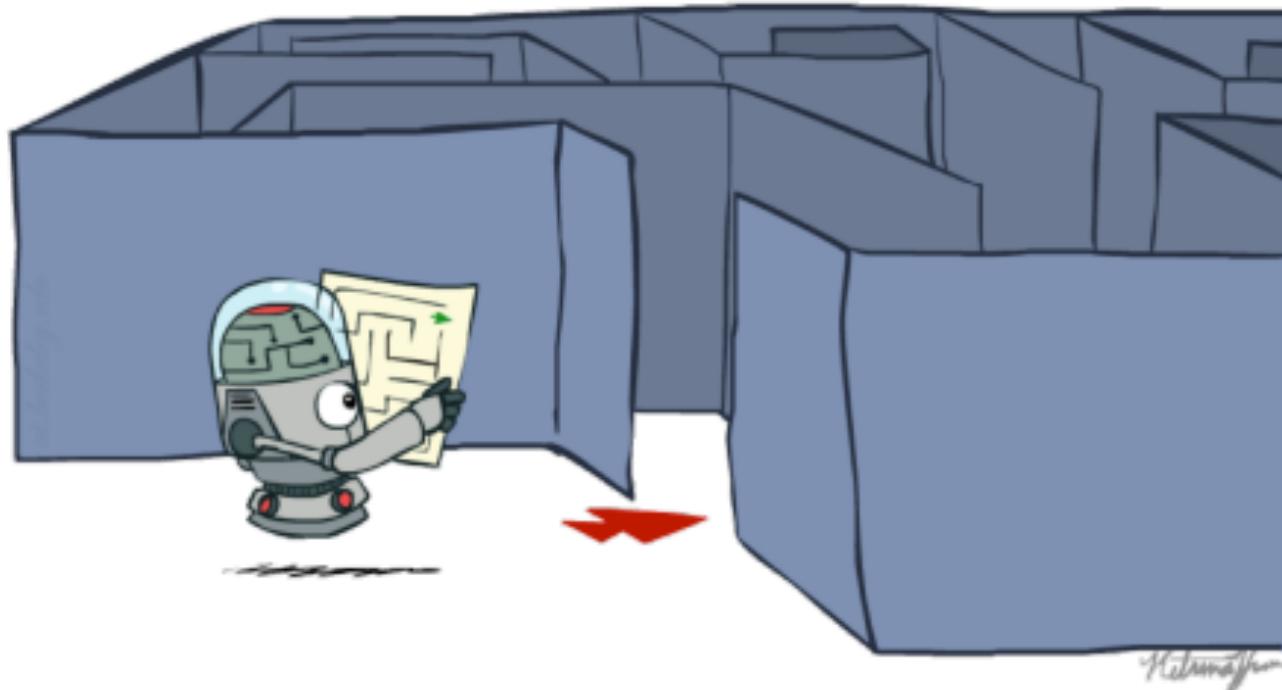


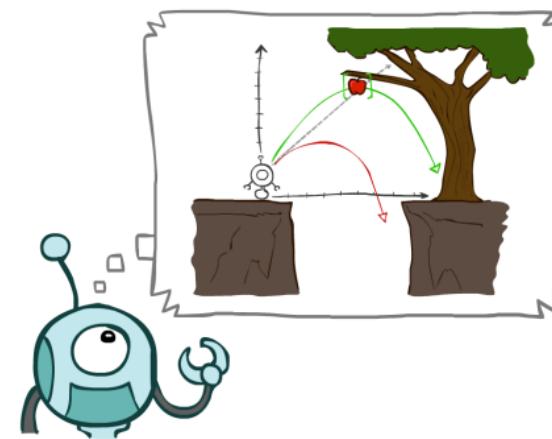
# Inteligencia Artificial **Solucionando Problemas por Búsqueda**



- Lectura: Capítulo 3 : Solving problems by Searching, “IA a modern approach.”

Se verá cómo un agente inteligente puede encontrar una secuencia de acciones que le permitirán llevar a cabo la consecución de un objetivo cuando no es posible hacerlo con una sola acción

- Los agentes reactivos no operan bien en entornos complejos. Las tablas de acción o las construcciones condición-acción son demasiado grandes o difíciles de aprender.



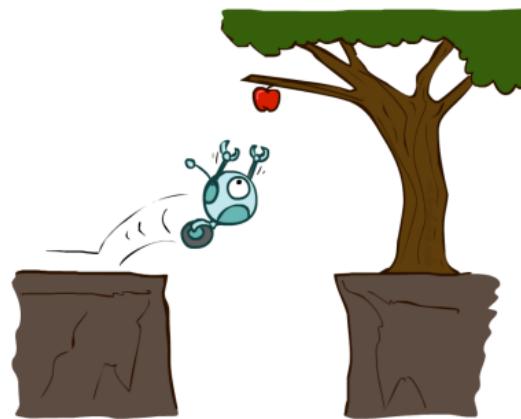
- Un agente basado en objetivos es más apropiado, ya que tienen en cuenta futuras acciones y sus efectos.

En este tema presentaremos los **agentes de resolución de problemas basados en búsquedas**

- Representación atómica (factored o structured es cosa de *agentes de planificación*)
- Uninformed vs Informed
- La solución siempre es una secuencia fija de acciones

**Se verá cómo un agente inteligente puede encontrar una secuencia de acciones que le permitirán llevar a cabo la consecución de un objetivo cuando no es posible hacerlo con una sola acción**

- Los agentes reactivos no operan bien en entornos complejos. Las tablas de acción o las construcciones condición-acción son demasiado grandes o difíciles de aprender.



- Un agente basado en objetivos es más apropiado, ya que tienen en cuenta futuras acciones y sus efectos.

*En este tema presentaremos los **agentes de resolución de problemas basados en búsquedas***

- Representación atómica (factored o structured es cosa de *agentes de planificación*)
- Uninformed vs Informed
- La solución siempre es una secuencia fija de acciones

# Agentes de Resolución de Problemas



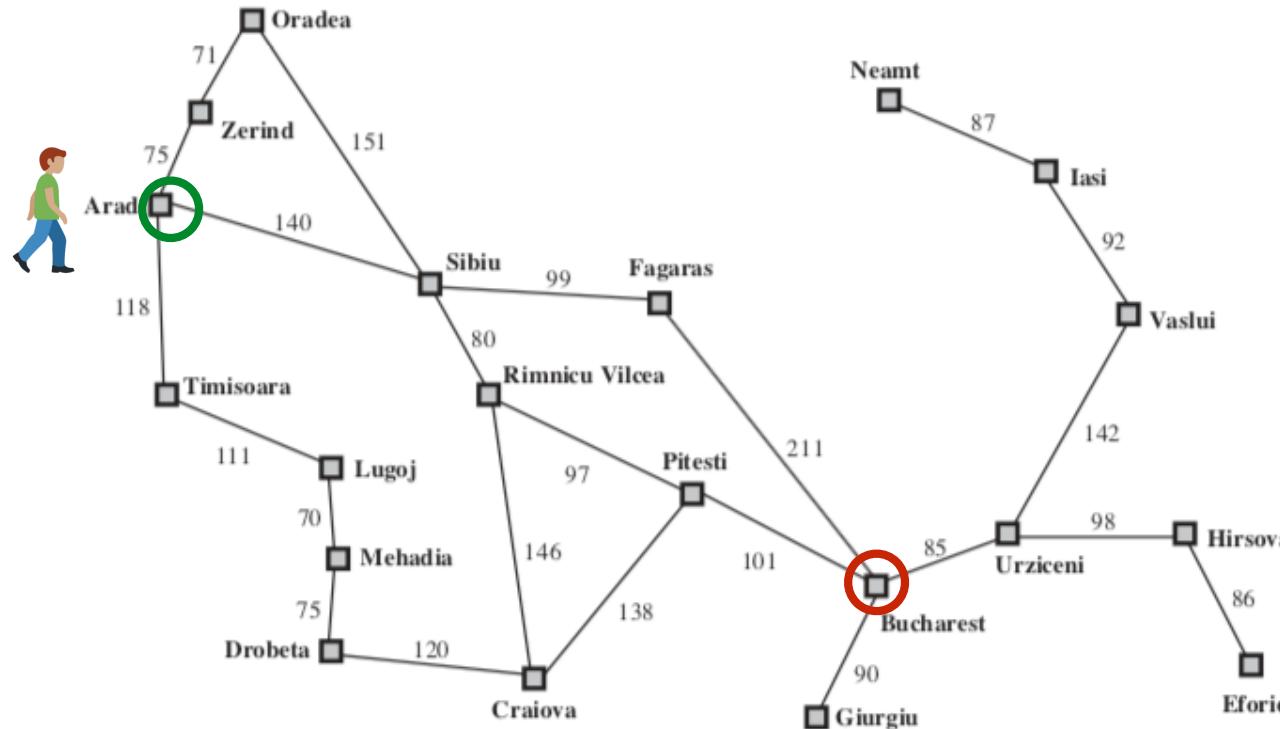
## Fases para la resolución de problemas, por parte de un Agente de resolución de problemas, en un entorno informado:

- **Formulación de objetivos**, basada en la situación actual, y la medida de desempeño del agente, es el primer paso en la resolución de problemas.
  - Los **objetivos** (o metas) ayudan a organizar el comportamiento al limitar los objetivos que el agente está tratando de lograr y, por lo tanto, las acciones que debe considerar. Ejemplo : Alcanzar la ciudad de Bucarest.
  - Un objetivo es satisfecho cuando se dan un conjunto de estados deseado.
- **Formulación del problema**, es el proceso de decidir qué acciones y estados considerar para dar por alcanzado el objetivo. Es un modelo abstracto de la parte relevante del mundo en el cual el agente se va a mover. Ejemplo: Moverse a ciudades que son alcanzables desde la que nos encontramos.
- **Búsqueda**: Antes de realizar cualquier acción en el mundo real, el agente simula secuencias de acciones en su modelo, buscando hasta que encuentra la secuencia de acciones necesarias para alcanzar el objetivo.
- **Ejecución**: El agente puede ejecutar las acciones encontradas en la búsqueda y que conducen al objetivo, caso de que estas existan.

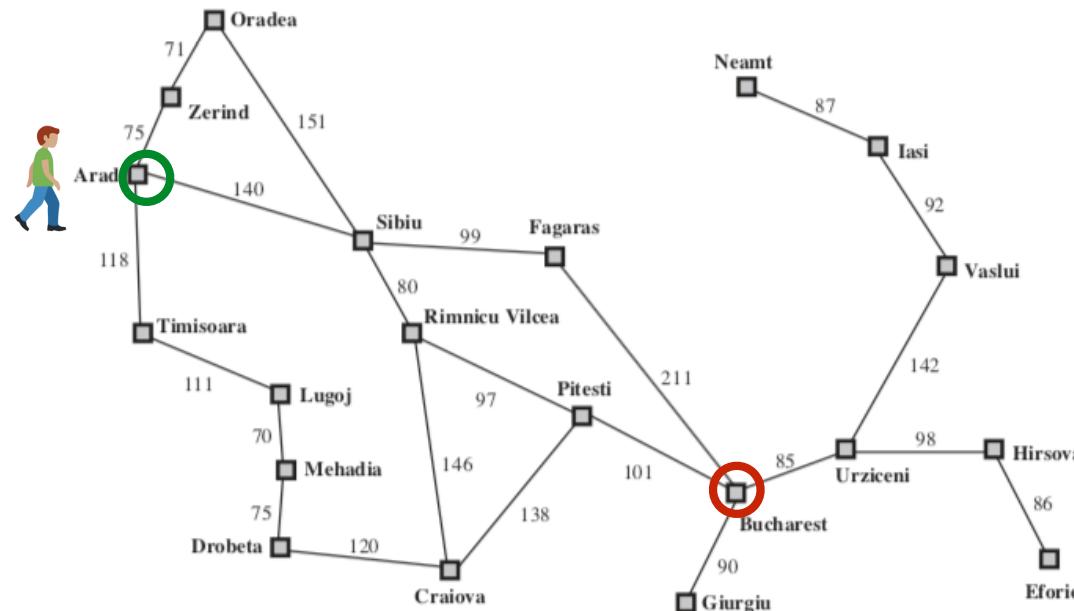


Intelligent  
Robotics  
Lab

*En general, un agente con varias opciones inmediatas de valor desconocido puede decidir qué hacer examinando primero acciones futuras que eventualmente conduzcan a estados de valor conocido.*



- El entorno tiene las siguientes propiedades:
  - **observable**
  - **discreto**
  - **conocido**
  - **determinista**
- El proceso de buscar una secuencia de acciones que alcance la meta se llama **búsqueda**.
- Un algoritmo de búsqueda toma un problema como entrada y devuelve una **solución** en forma de secuencia de acciones.
- Una vez que se encuentra una solución, se pueden llevar a cabo las acciones que recomienda. Esto es la **fase de ejecución**.
- En teoría de control : *Lazo abierto* (ignora las percepciones) vs *Lazo cerrado* (*Las tiene en cuenta*)



- El entorno tiene las siguientes propiedades:
  - **observable**
  - **discreto**
  - **conocido**
  - **determinista**
- El proceso de buscar una secuencia de acciones que alcance la meta se llama **búsqueda**.
- Un algoritmo de búsqueda toma un problema como entrada y devuelve una **solución** en forma de secuencia de acciones.
- Una vez que se encuentra una solución, se pueden llevar a cabo las acciones que recomienda. Esto es la **fase de ejecución**.
- En teoría de control : *Lazo abierto* (ignora las percepciones) vs *Lazo cerrado* (*Las tiene en cuenta*)

```

function SIMPLE-PROBLEM-SOLVING-AGENT(percept) returns an action
  persistent: seq, an action sequence, initially empty
  state, some description of the current world state
  goal, a goal, initially null
  problem, a problem formulation

  state  $\leftarrow$  UPDATE-STATE(state, percept)
  if seq is empty then
    goal  $\leftarrow$  FORMULATE-GOAL(state)
    problem  $\leftarrow$  FORMULATE-PROBLEM(state, goal)
    seq  $\leftarrow$  SEARCH(problem)
    if seq = failure then return a null action
    action  $\leftarrow$  FIRST(seq)
    seq  $\leftarrow$  REST(seq)
  return action

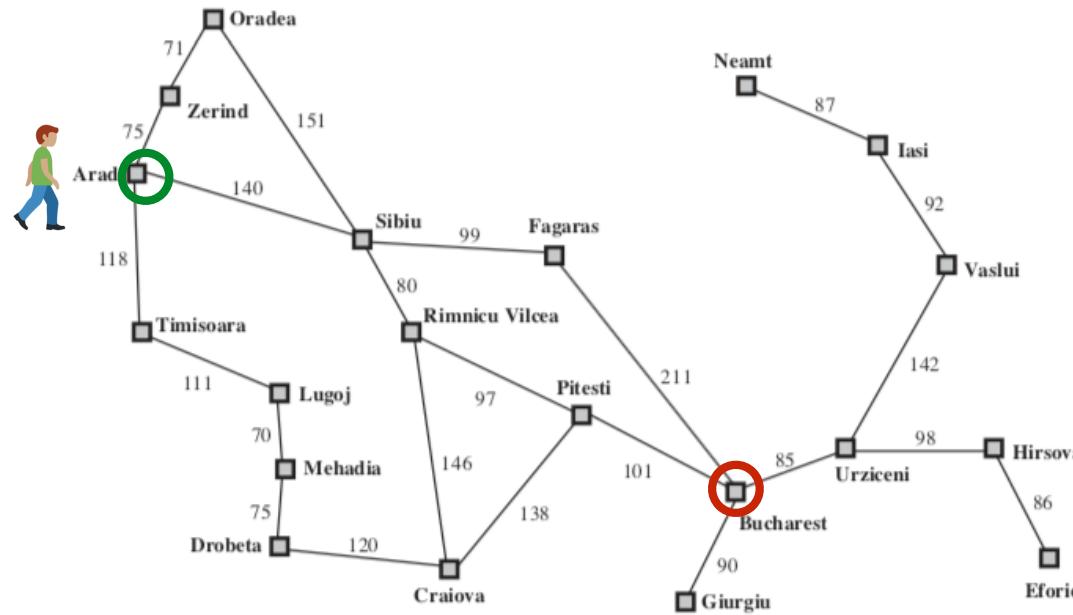
```

# Problemas de búsqueda bien definidos y soluciones

- Un **problema de búsqueda** se define por los siguientes componentes:
  1. El **estado inicial** del agente  $In(Arad)$
  2. Las posibles **acciones** del agente **aplicables** en cada estado

*ACTIONS(s)*

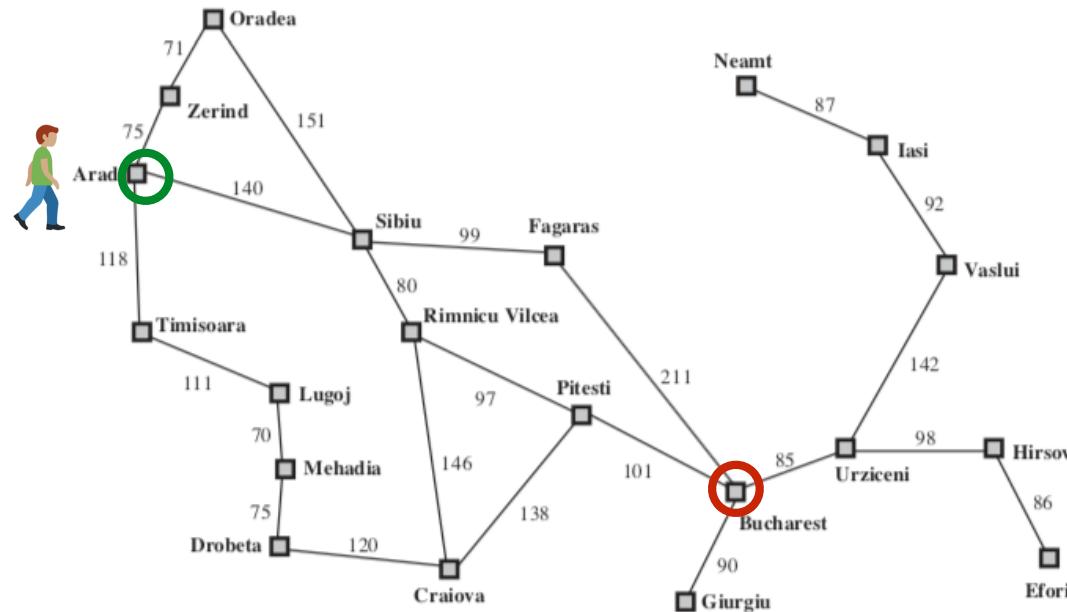
$$ACTIONS( In(Arad) ) = \{ Go(Sibiu), Go(Timisoara), Go(Zerind) \}$$



# Problemas de búsqueda bien definidos y soluciones

- Un problema de búsqueda se define por los siguientes componentes:
  1. El **estado inicial** del agente  $In(Arad)$
  2. Las posibles **acciones** del agente **aplicables** en cada estado
  3. Lo que hace cada acción - su **modelo de transiciones**
    - sucesor
    - estado inicial + acciones + modelo de transiciones = **espacio de estados del problema**
    - Grafo dirigido
      - Nodos = Estados
      - Transiciones = Acciones
      - Ruta = secuencia de estados conectados por una secuencia de acciones

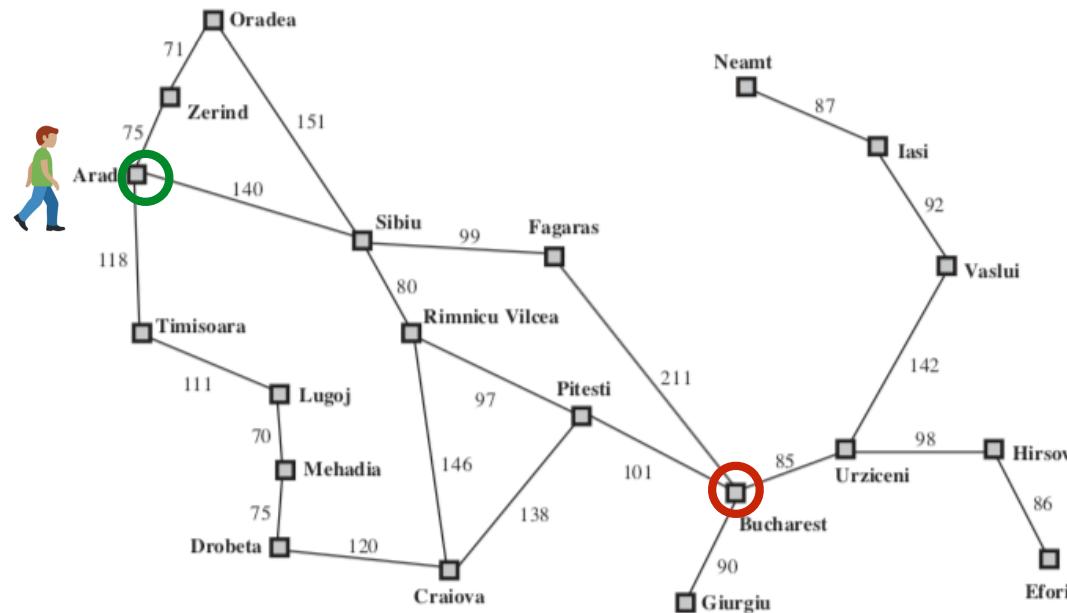
$$RESULT(s, a)$$

$$RESULT( In(Arad), Go(Zerind) ) = In(Zerind)$$


# Problemas de búsqueda bien definidos y soluciones

- Un problema de búsqueda se define por los siguientes componentes:
  1. El **estado inicial** del agente *In (Arad)*
  2. Las posibles **acciones** del agente **aplicables** en cada estado
  3. Lo que hace cada acción - su **modelo de transiciones**
    - **sucesor**
    - estados + acciones + modelo de transiciones = **espacio de estados del problema**
    - Grafo dirigido
      - Nodos = Estados
      - Transiciones = Acciones
      - Ruta = secuencia de estados conectados por una secuencia de acciones
  4. **Test de objetivo**

*{In(Bucharest)}*

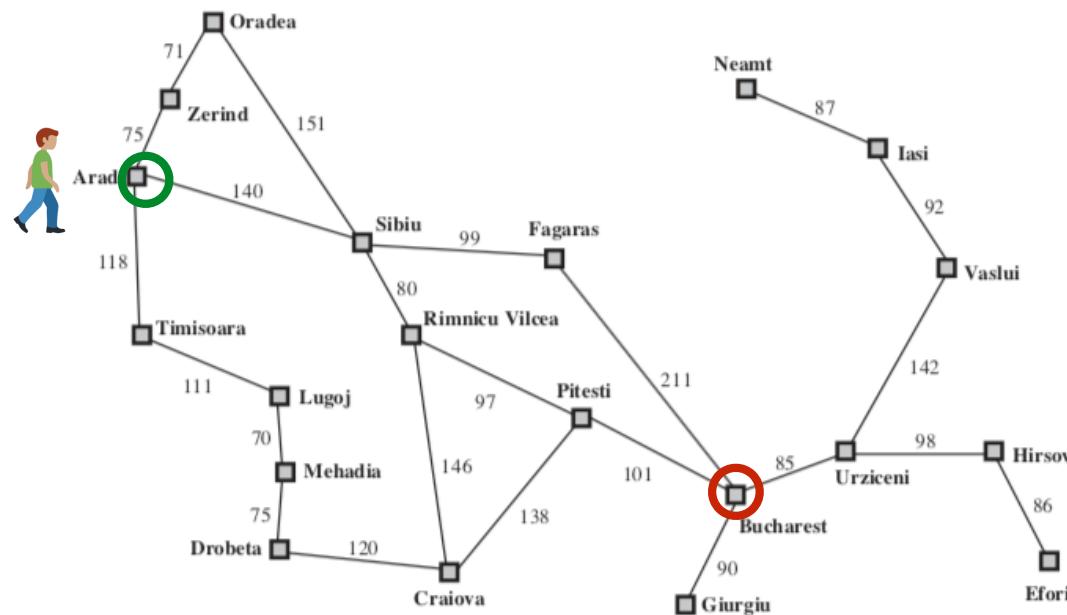


# Problemas de búsqueda bien definidos y soluciones

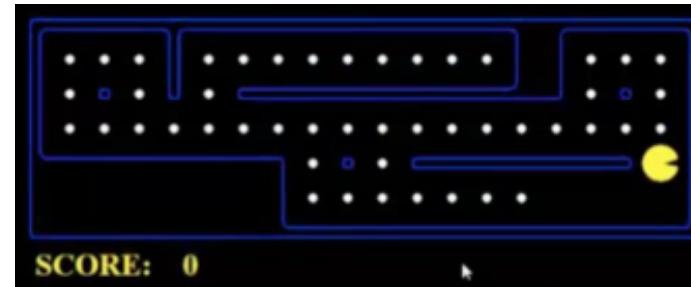
- Un **problema de búsqueda** se define por los siguientes componentes:

1. El **estado inicial** del agente *In (Arad)*
2. Las posibles **acciones** del agente **aplicables** en cada estado
3. Lo que hace cada acción - su **modelo de transiciones**
  - **sucesor**
  - estados + acciones + modelo de transiciones = **espacio de estados del problema**
  - Grafo dirigido
    - Nodos = Estados
    - Transiciones = Acciones
    - Ruta / Solución = secuencia de estados conectados por una secuencia de acciones
4. **Test de objetivo**
5. **Coste de la solución** — medida de rendimiento

$$\begin{aligned} \text{total\_cost} &= \sum_{i=0}^{n-1} \text{cost}(s_i, a_i, s_{i+1}) \\ \text{path} &= \{s_0, a_0, s_1, a_1, \dots, a_{n-1}, s_n\} \end{aligned}$$



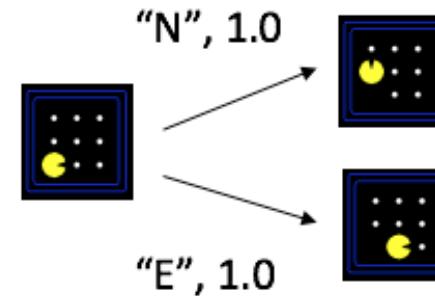
## Problemas bien definidos y soluciones



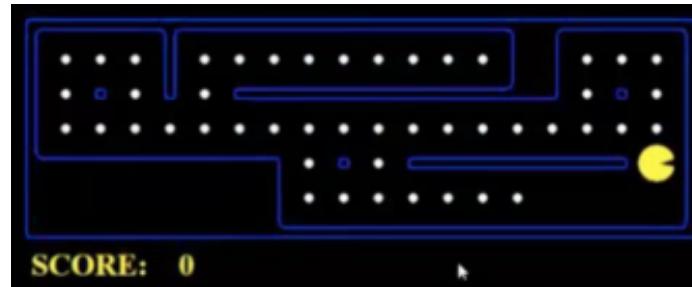
Espacio de estados



Acciones y modelo de transición



## Problemas bien definidos y soluciones



### Problema: Encontrar una ruta

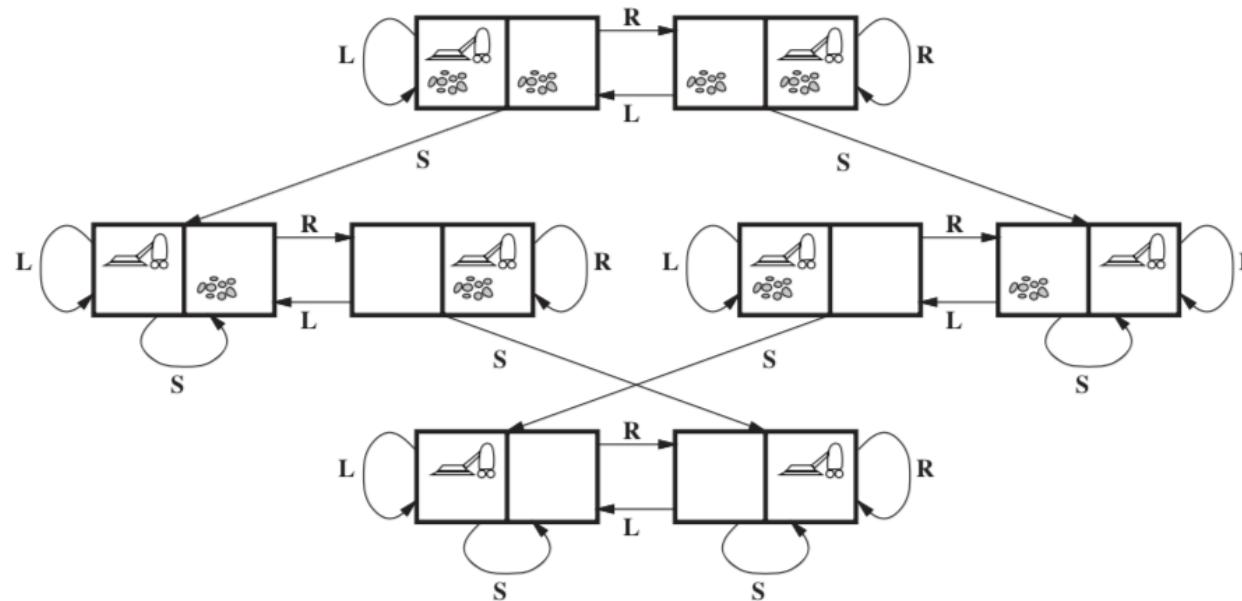
- Estados:  $(x, y)$
- Acciones: NSEW
- Sucesor: Actualizar posición
- Test de objetivo:  $(x, y) == \text{end}$

### Problema: Comer puntos

- Estados:  $\{(x, y), \text{puntos: booleanos}\}$
- Acciones: NSEW
- Sucesor: Actualizar posición y los puntos
- Test de objetivo: todos los puntos a false

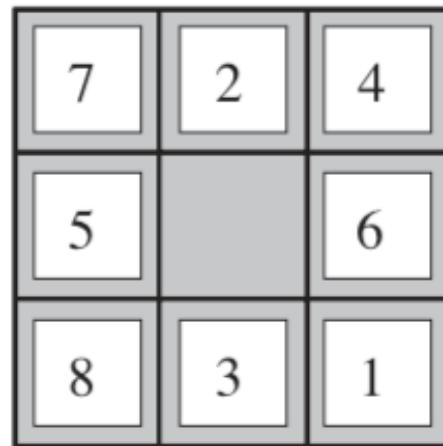
# Problemas de ejemplo

# Mundo cuadriculado

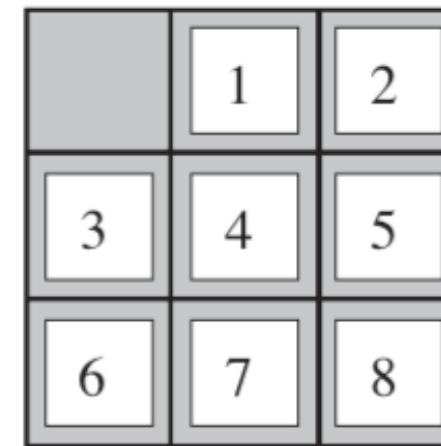


- Estados:**
  - Depende de dónde está el agente y el estado de la localización. Agente + suciedad.
  - 8 estados para dos localizaciones
- Estado Inicial:** cualquiera
- Acciones:** Left, Right, Suck
- Modelo de Transición:**
- Test de Objetivo:** Chequea si todas las localizaciones están limpias
- Coste de la solución:** coste de 1 por cada paso en la solución

## Rompecabezas de ficha deslizante



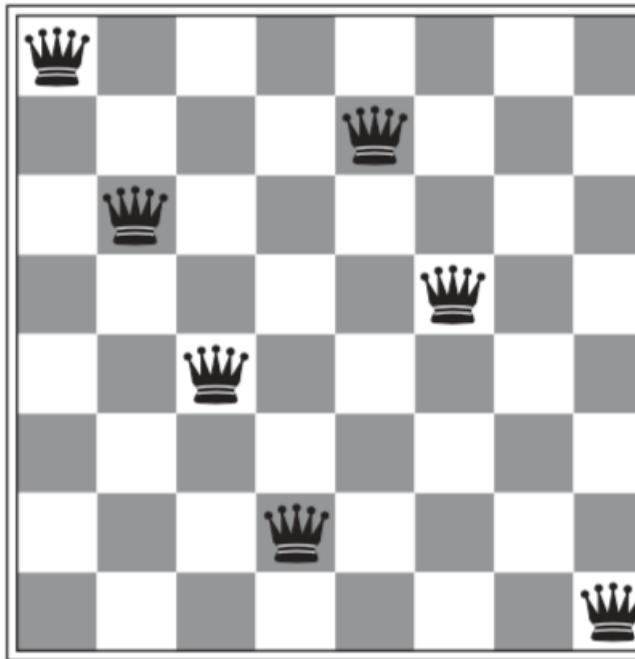
Start State



Goal State

1. **Estados:** la localización de cada una de las 8 fichas y del espacio vacío
  2. **Estado Inicial:** cualquiera
  3. **Acciones:** Mover el espacio vacío con *Left, Right, Up, Down*
  4. **Modelo de Transición:** Dado un estado y una acción, se intercambian las posiciones el espacio en blanco y la ficha
  5. **Test de Objetivo:** Chequea si la posición de las fichas están colocadas
  6. **Coste de la solución:** coste de 1 por cada paso en la solución
- 
- De la familia de rompecabezas de bloques deslizantes, usado como problemas de prueba para nuevos algoritmos de búsqueda en IA.
  - NP-completo
  - $9!/2 = 181,440$  estados alcanzables

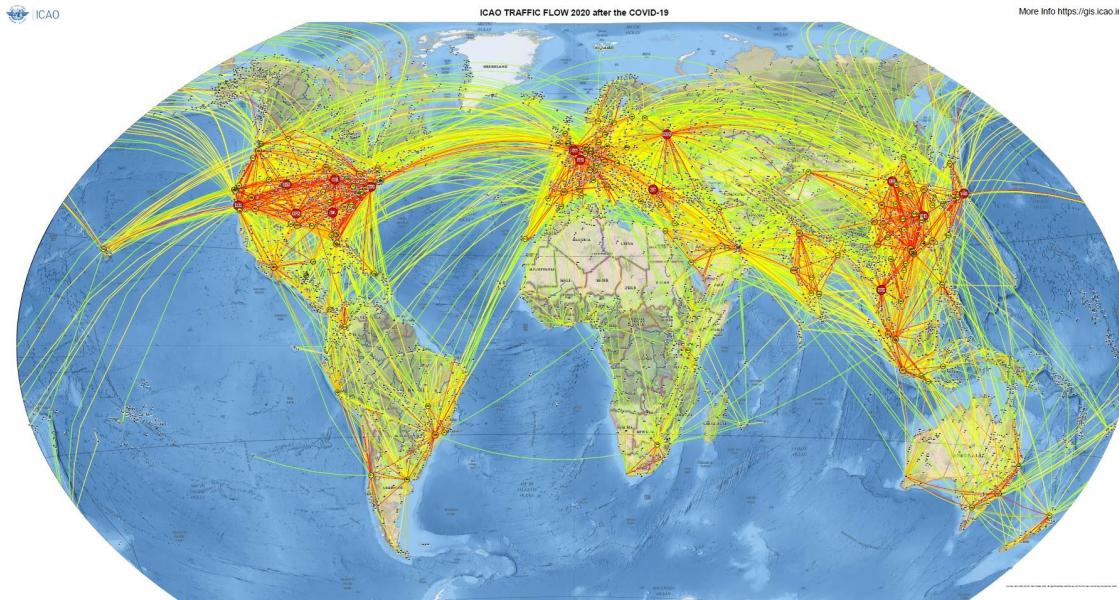
## 8 Reinas



1. **Estados:** Cualquier disposición de 0 a 8 reinas en el tablero es un estado, una por columna en la columna de más a la izquierda sin que se ataque
2. **Estado Inicial:** Tablero vacío
3. **Acciones:** Añadir una reina a una casilla vacía en la columna de más a la izquierda sin que se ataquen
4. **Modelo de Transición:** Función que devuelve el tablero con la reina añadida
5. **Test de Objetivo:** Las 8 reinas en el tablero están colocadas y ninguna se ataca
6. **Coste de la solución:** coste de 1 por cada paso en la solución

2057 posibles secuencias a investigar.

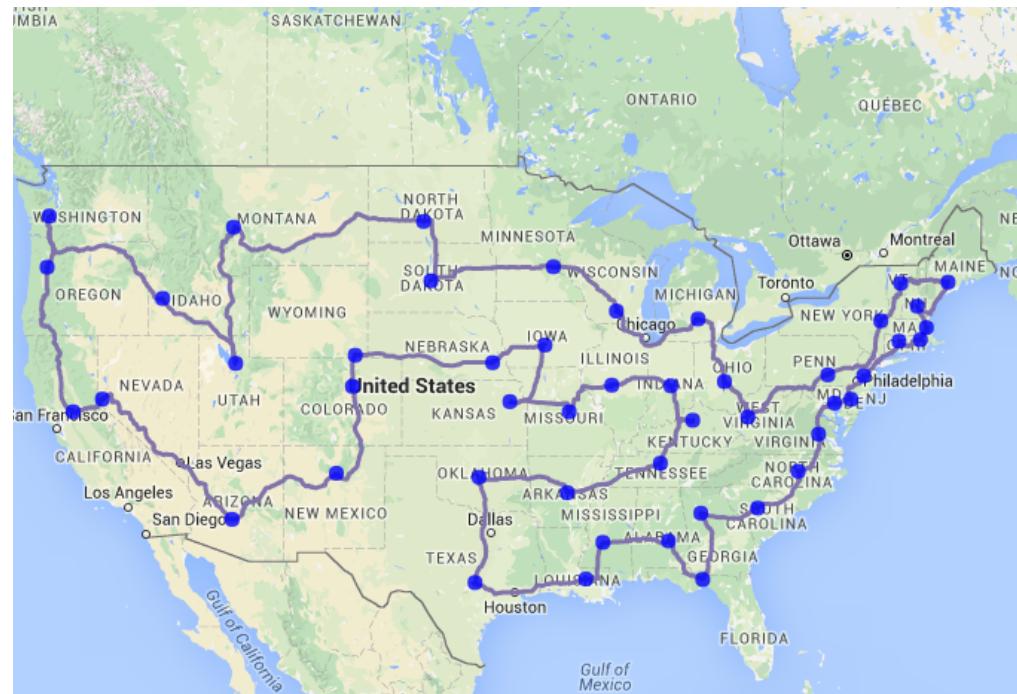
## Problemas reales



- Problemas de cálculo de rutas
- Web sites, GPS....
- **Sistemas de planificación de viajes de aerolíneas**

1. **Estados:** Una localización y la hora. Si el coste de una acción depende de las acciones anteriores, los estados deben almacenar información extra
2. **Estado Inicial:** Especificado por la consulta inicial del usuario
3. **Acciones:** Tomar un vuelo en la localización actual después del tiempo actual
4. **Modelo de Transición:** El estado resultante (nueva localización y tiempo)
5. **Test de Objetivo:** Si está en el destino final
6. **Coste de la solución:** dinero, tiempo, tiempo de espera, calidad del asiento, recompensa...

# Problemas reales



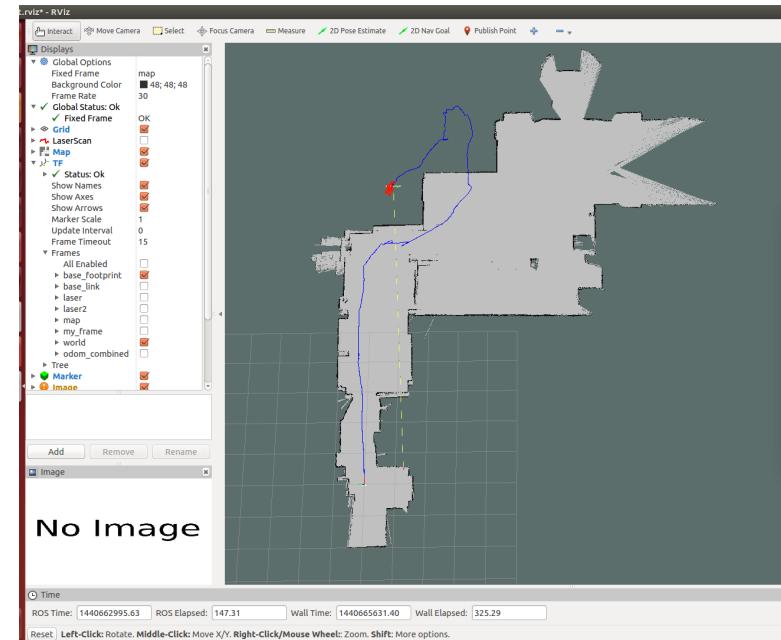
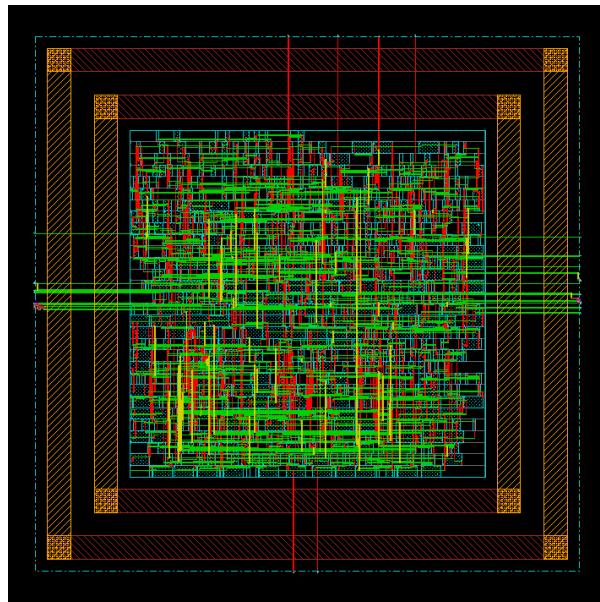
- **Problemas de gira:** Visitar todas las ciudades de un país, al menos una vez, terminando en la ciudad inicial

*In(Vaslui), Visited({Bucharest, Urziceni, Vaslui})*

- **Problemas del vendedor ambulante:** Visitar todas las ciudades de un país, sin repetir, recorriendo la menor cantidad de kilómetros

- Planificación de atornillamiento
  - Robots reponedores

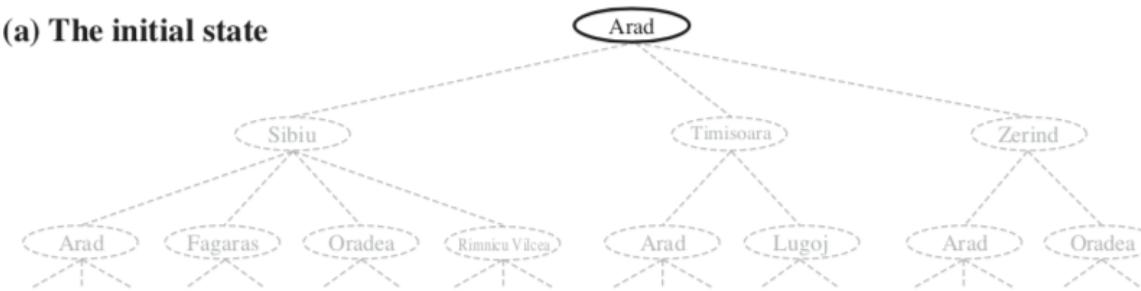
## Problemas reales



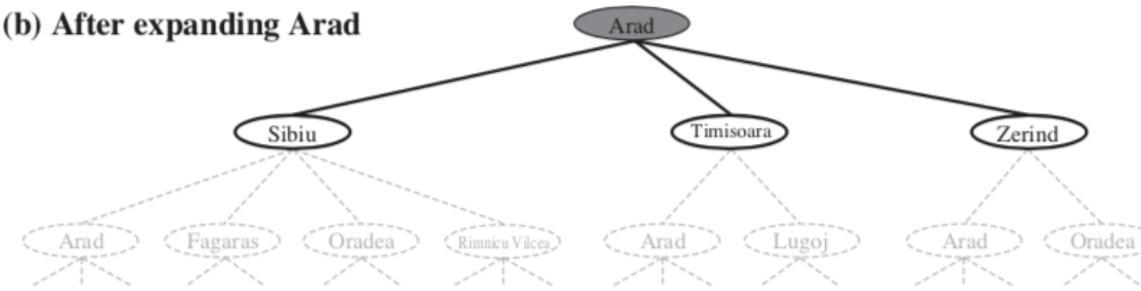
# Buscando soluciones

- Una vez definido un problema, solucionemos
- **Árbol de búsqueda**
- **Nodos** - estado en el espacio de estados
- **Expansión** de un **nodo padre** para generar **nodos hijo mediante acciones disponibles**

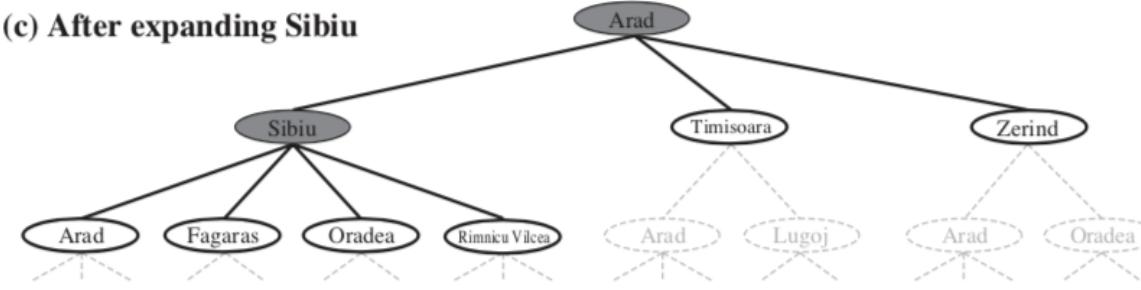
(a) The initial state



(b) After expanding Arad



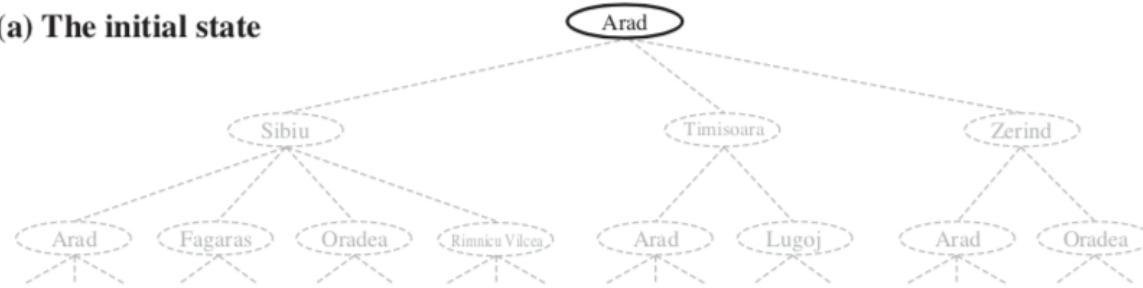
(c) After expanding Sibiu



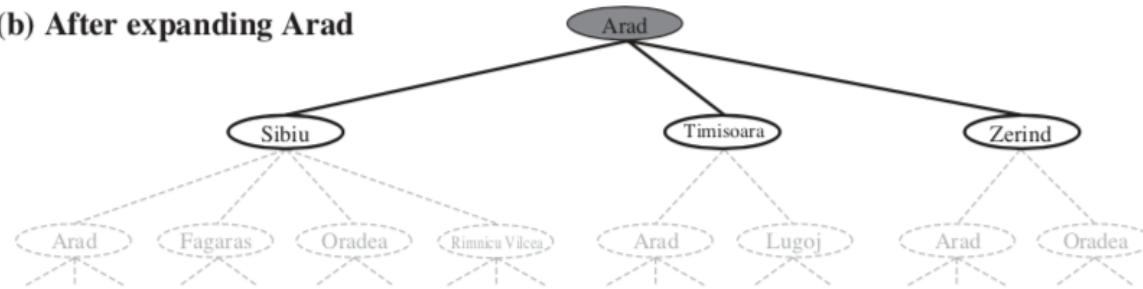
- **Nodos hoja**
- Los nodos hoja que se pueden expandir se llaman **frontera (lista abierta)**
- La **estrategia de búsqueda** define como expandir los nodos frontera

Esta es la esencia de la búsqueda: seguir una opción ahora y dejar las otras a un lado para más adelante, en caso de que la primera opción no conduzca a una solución.

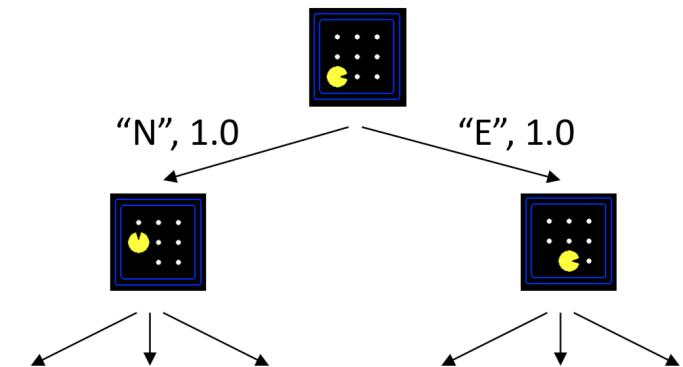
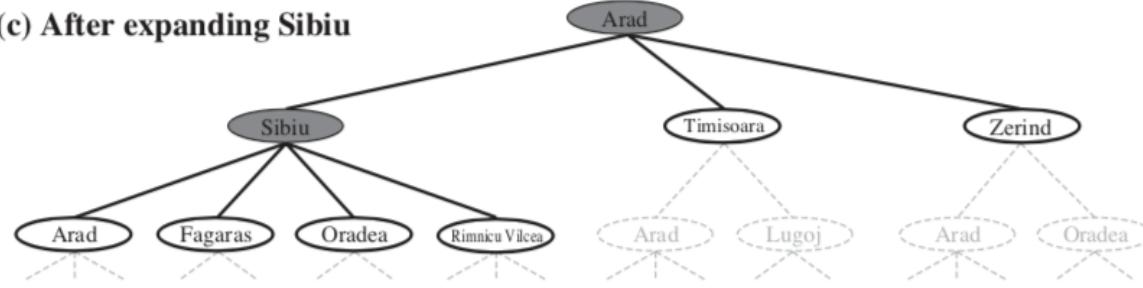
(a) The initial state



(b) After expanding Arad



(c) After expanding Sibiu



- **Nodos hoja**
- Los nodos hoja que se pueden expandir se llaman **frontera (lista abierta)**
- La **estrategia de búsqueda** define como expandir los nodos frontera
- Evitar bucles con estados repetidos. En rejillas rectangulares es complicado evitar los **caminos redundantes**

Esta es la esencia de la búsqueda: seguir una opción ahora y dejar las otras a un lado para más adelante, en caso de que la primera opción no conduzca a una solución.

**function** TREE-SEARCH(*problem*) **returns** a solution, or failure  
  initialize the frontier using the initial state of *problem*

**loop do**

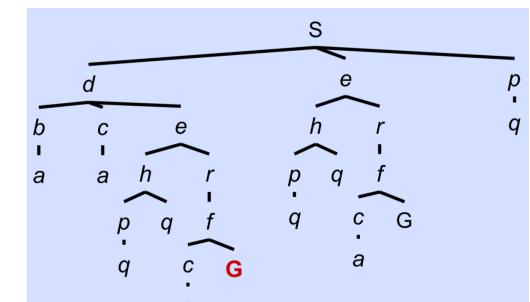
**if** the frontier is empty **then return** failure

  choose a leaf node and remove it from the frontier

**if** the node contains a goal state **then return** the corresponding solution

  expand the chosen node, adding the resulting nodes to the frontier

(Recuerda, es el punto que apunta a sus padres)



- **Nodos hoja**
- Los nodos hoja que se pueden expandir se llaman **frontera (lista abierta)**
- La **estrategia de búsqueda** define como expandir los nodos frontera
- Evitar bucles con estados repetidos. En rejillas rectangulares es complicado evitar los **caminos redundantes**
- El **conjunto explorado (lista cerrada)** contiene los estados ya explorados

*Los algoritmos que olvidan su historia están condenados a repetirla*

**function** GRAPH-SEARCH(*problem*) **returns** a solution, or failure

  initialize the frontier using the initial state of *problem*

*initialize the explored set to be empty*

**loop do**

**if** the frontier is empty **then return** failure

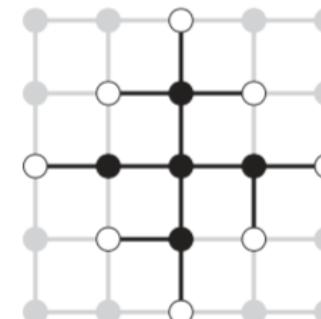
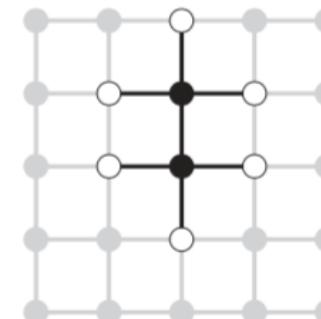
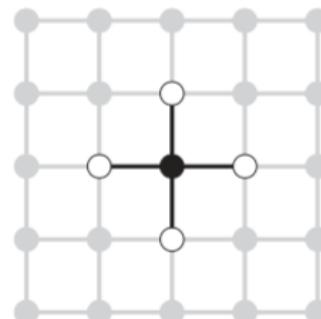
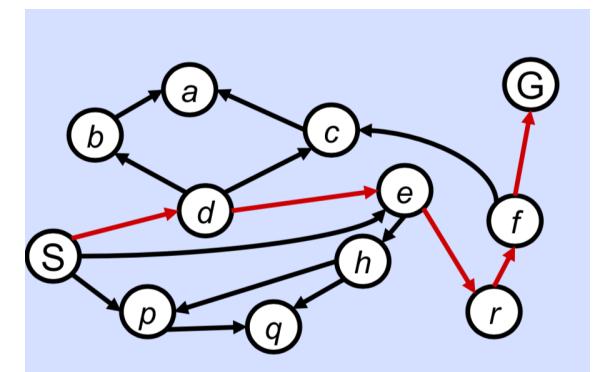
    choose a leaf node and remove it from the frontier

**if** the node contains a goal state **then return** the corresponding solution

*add the node to the explored set*

    expand the chosen node, adding the resulting nodes to the frontier

*only if not in the frontier or explored set*



ES EL TEMPLATE

## Best First Search

↓  
primos los de  
menor número

(cola con prioridad)

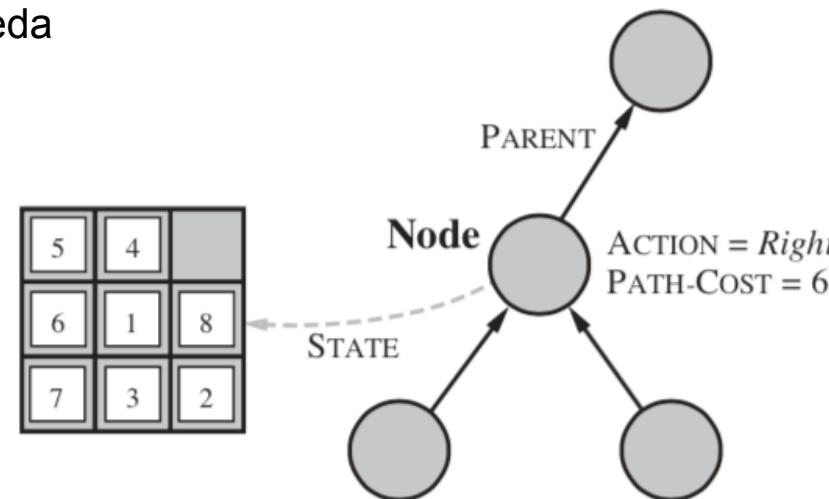
```
function BEST-FIRST-SEARCH(problem,f) returns a solution node or failure
    node  $\leftarrow$  NODE(STATE=problem.INITIAL)
    frontier  $\leftarrow$  a priority queue ordered by f, with node as an element
    reached  $\leftarrow$  a lookup table, with one entry with key problem.INITIAL and value node
    while not Is-EMPTY(frontier) do
        node  $\leftarrow$  POP(frontier)
        if problem.IS-GOAL(node.STATE) then return node
        for each child in EXPAND(problem, node) do
            s  $\leftarrow$  child.STATE
            if s is not in reached or child.PATH-COST < reached[s].PATH-COST then
                reached[s]  $\leftarrow$  child
                add child to frontier
    return failure
```

Comprobación tardía de objetivo

```
function EXPAND(problem, node) yields nodes
    s  $\leftarrow$  node.STATE
    for each action in problem.ACTIONS(s) do
        s'  $\leftarrow$  problem.RESULT(s, action)
        cost  $\leftarrow$  node.PATH-COST + problem.ACTION-COST(s, action, s')
        yield NODE(STATE=s', PARENT=node, ACTION=action, PATH-COST=cost)
```

# Infraestructura de los algoritmos de búsqueda

- Los algoritmos de búsqueda requieren una estructura de datos para seguir la pista del árbol de búsqueda
- Por cada nodo  $n$ 
  - $n.\text{STATE}$
  - $n.\text{PARENT}$
  - $n.\text{ACTION}$
  - $n.\text{PATH-COST}$  (aka  $g(n)$ )



- A partir de un nodo padre, obtener un nodo hijo es fácil:

```
function CHILD-NODE(problem, parent, action) returns a node
    return a node with
```

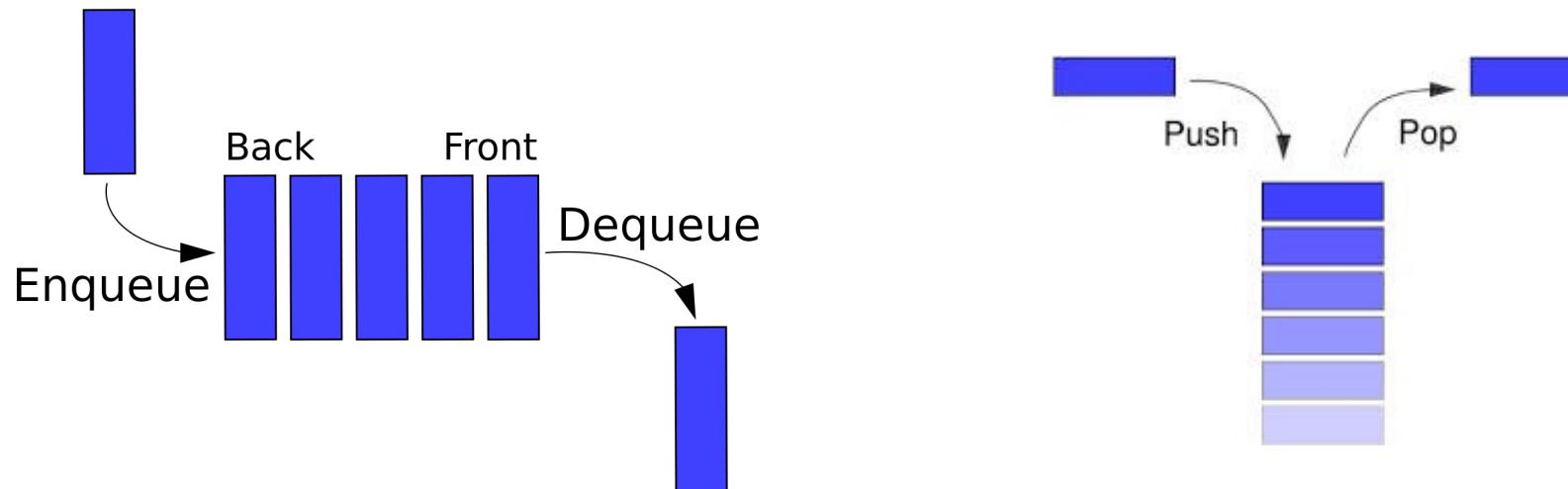
STATE = *problem.RESULT(parent.STATE, action)*,

PARENT = *parent*, ACTION = *action*,

PATH-COST = *parent.PATH-COST + problem.STEP-COST(parent.STATE, action)*

# Infraestructura de los algoritmos de búsqueda

- Los nodos frontera se introducen en una cola para saber cuál es el siguiente en expandir
  - FIFO
  - LIFO (pila)
  - de **prioridad**
- Operaciones:
  - EMPTY? (q)
  - POP (q)
  - INSERT(e, q)
- El conjunto de exploración puede estar en una tabla hash, para comprobar rápido

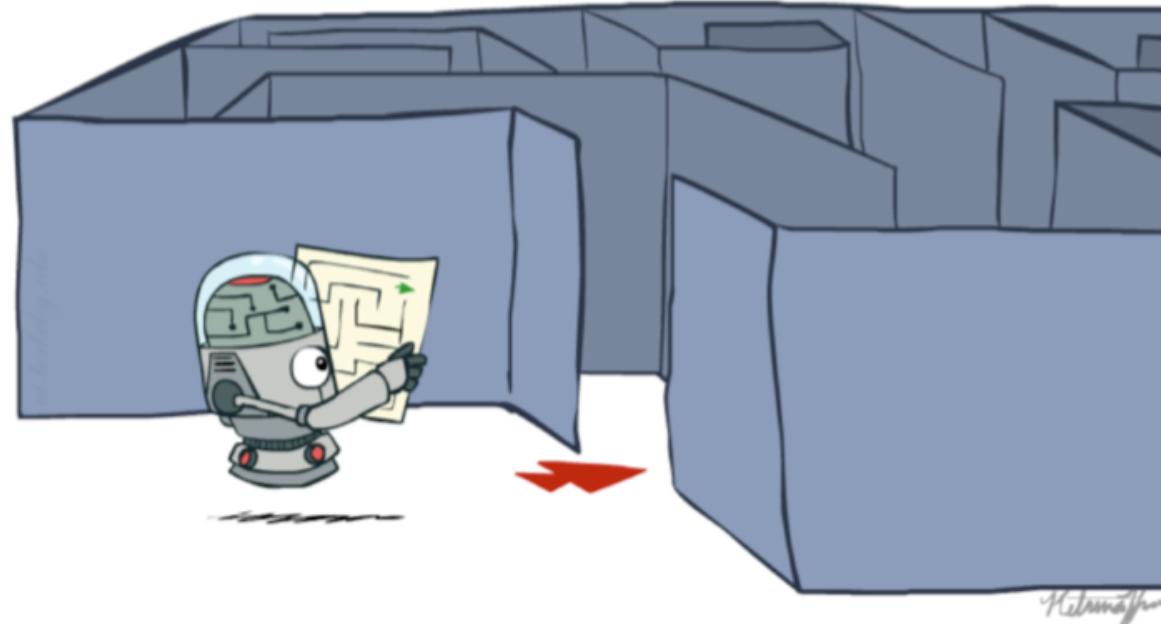


## Midiendo el rendimiento

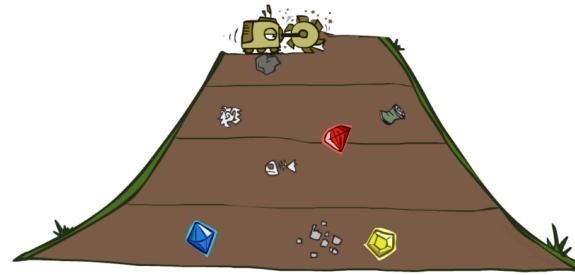
- **Completitud:** ¿Está garantizado el algoritmo para encontrar una solución cuando la hay?
  - **Optimidad:** ¿La estrategia encuentra la solución óptima?
  - **Complejidad en tiempo:** ¿Cuánto tiempo se tarda en encontrar una solución?
  - **Complejidad en espacio:** ¿Cuánta memoria se necesita para realizar la búsqueda?
- 
- Un grafo de estas de búsqueda puede medirse como  $|V| + |E|$
  - Puede ser infinito, por lo que se usa
    - el factor de ramificación  $b$
    - la profundidad  $d$
    - la longitud máxima  $m$
- 
- La efectividad de un algoritmo se puede medir:
    - el coste de la búsqueda — complejidad y memoria
    - el coste total — coste de búsqueda + coste de la solución

# Estrategias de búsqueda no informadas

- Búsqueda no informada, a.k.a. **búsqueda ciega**
- Las estrategias no tienen información adicional sobre los estados más allá de la proporcionada en la definición del problema.
- **El algoritmo no sabe lo cerca que puede estar el nodo objetivo.**
- **Generar sucesores** y distingue un estado objetivo de un estado no objetivo.
- Las **estrategias se distinguen por el orden en que se expanden los nodos.**
- Las estrategias que saben si un estado sin objetivo es "más prometedor" que otro se denominan búsqueda informada o estrategias de búsqueda heurística. Se verá en el próximo apartado.



## Búsqueda en amplitud primero (breadth-first search)



- El nodo raíz se expande primero, luego todos los sucesores del nodo raíz se expanden a continuación, luego sus sucesores, y así sucesivamente.
- En general, todos los nodos se expanden a una profundidad determinada en el árbol de búsqueda antes de que se expandan los nodos del siguiente nivel
- Cola FIFO en la frontera
- El test de objetivo conseguido se aplica a cada nodo cuando se genera en lugar de cuando se selecciona para la expansión

expandió - mutava frontera  
temprano →  
tardío → expandir



# Búsqueda en amplitud primero (breadth-first search)

Tiene una complejidad exponencial  $O(b^d)$

Es un grafo de computación (comprueba)

```

function BREADTH-FIRST-SEARCH(problem) returns a solution node or failure
    node  $\leftarrow$  NODE(problem.INITIAL)
    if problem.IS-GOAL(node.STATE) then return node
    frontier  $\leftarrow$  a FIFO queue, with node as an element
    reached  $\leftarrow \{problem.INITIAL\}
    while not IS-EMPTY(frontier) do           Cola FIFO
        node  $\leftarrow$  POP(frontier)
        for each child in EXPAND(problem, node) do
            s  $\leftarrow$  child.STATE
            if problem.IS-GOAL(s) then return child
            if s is not in reached then
                add s to reached
                add child to frontier
    return failure                                Comprobación temprana de objetivo$ 
```

# Búsqueda en amplitud primero (breadth-first search)

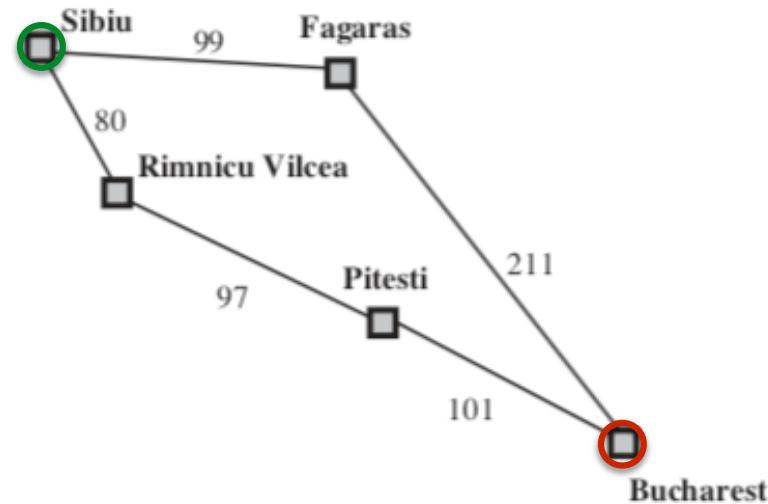
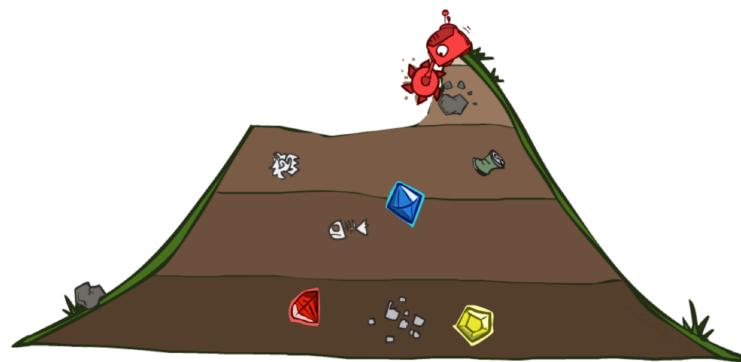
- Es completo.
- No es óptimo, salvo si el coste de cada acción es idéntico, o es una función no decreciente de la profundidad del nodo.
- Es costoso en tiempo y espacio.

$$b + b^2 + b^3 + \cdots + b^d = O(b^d)$$

Depth	Nodes	Time	Memory
2	110	.11 milliseconds	107 kilobytes
4	11,110	11 milliseconds	10.6 megabytes
6	$10^6$	1.1 seconds	1 gigabyte
8	$10^8$	2 minutes	103 gigabytes
10	$10^{10}$	3 hours	10 terabytes
12	$10^{12}$	13 days	1 petabyte
14	$10^{14}$	3.5 years	99 petabytes
16	$10^{16}$	350 years	10 exabytes

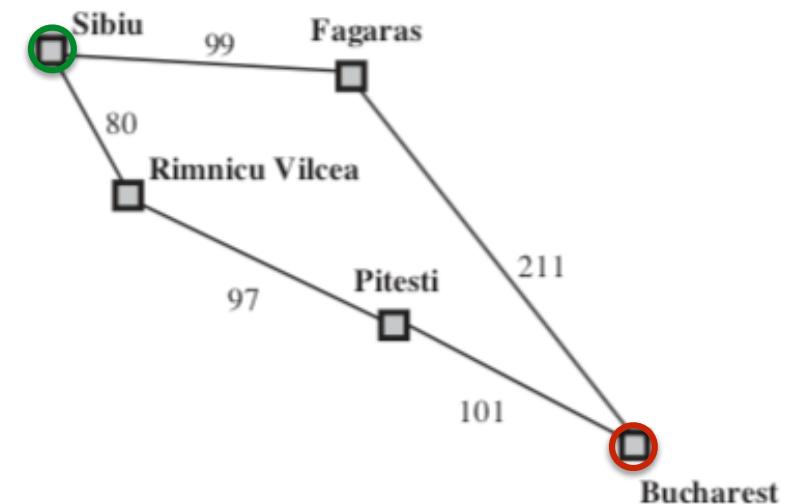
## Búsqueda en coste uniforme (uniform-cost search)

- En lugar de expandir el nodo más cercano, la búsqueda de coste uniforme **expande el nodo  $n$  con el coste de ruta más bajo  $g(n)$ .**
- Esto se hace almacenando la frontera como una cola de prioridad ordenada por  $g$ .
- El test de objetivo alcanzado se aplica a un nodo cuando se selecciona para expansión en lugar de cuando se genera por primera vez.
- Se agrega un test en caso de que se encuentre una ruta mejor para un nodo actualmente en la frontera.



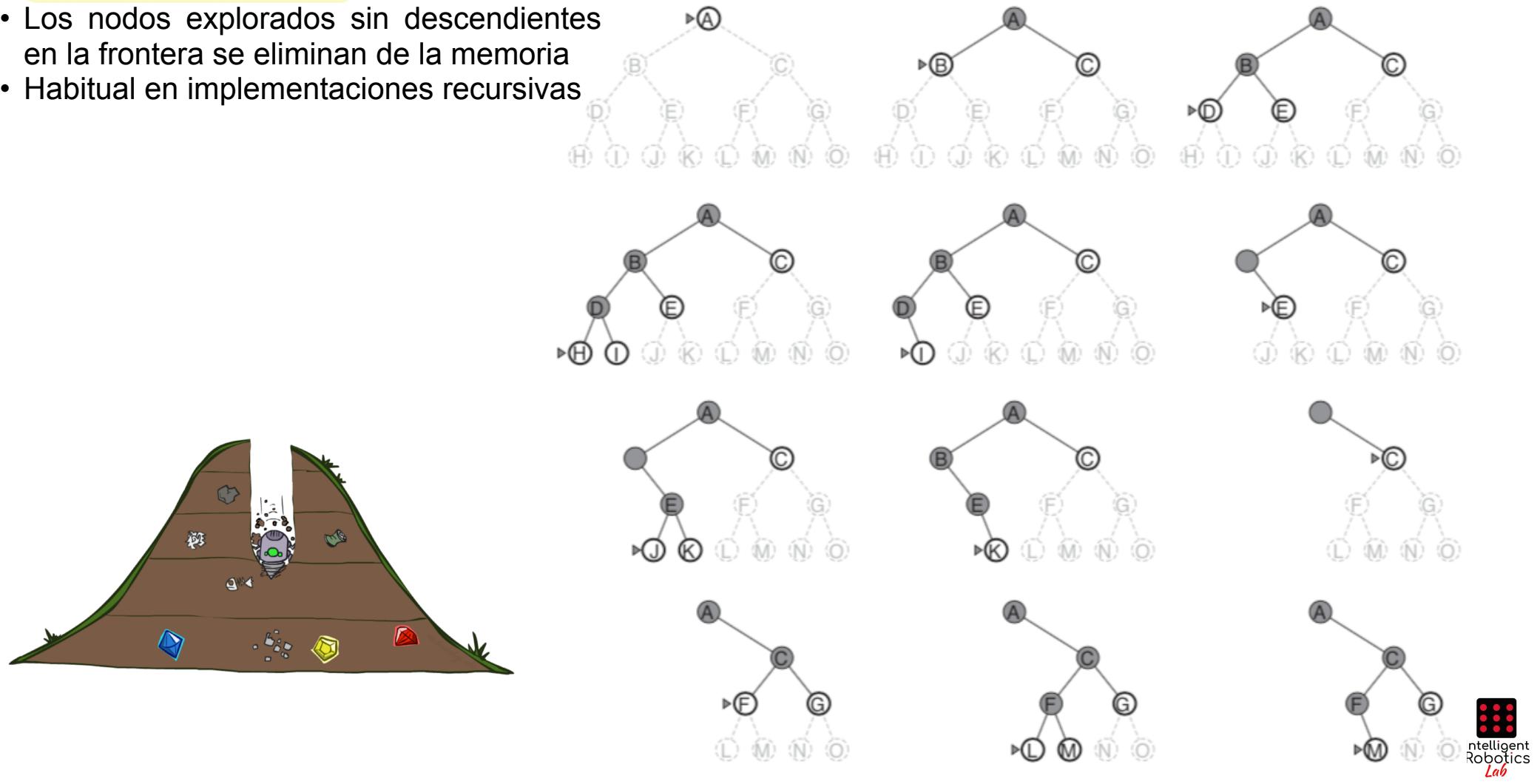
# Búsqueda en coste uniforme (uniform-cost search)

```
function UNIFORM-COST-SEARCH(problem) returns a solution node, or failure
  return BEST-FIRST-SEARCH(problem, PATH-COST)
```



## Búsqueda en profundidad primero (depth-first search)

- Expande el nodo más profundo en la frontera actual del árbol de búsqueda.
- Cola LIFO en la frontera
- Los nodos explorados sin descendientes en la frontera se eliminan de la memoria
- Habitual en implementaciones recursivas



# Búsqueda en profundidad primero (depth-first search)

- Es completo en graph-search version si el número de estados es finito
  - No es completo en tree-search version, ya que se pueden crear bucles infinitos
    - Se puede añadir un test al añadir nuevos estados si no están en el camino a la raíz, pero podría generar rutas redundantes
  - En espacios de estados infinitos, ambas versiones fallan si se sigue una ruta equivocada
  - Es sub-óptimo, ya que explora primero aunque el objetivo esté en la frontera
- 
- Si es tan malo ¿por qué usarlo? en tree-search version tiene mejor complejidad espacial que la búsqueda en amplitud

# Búsqueda en profundidad limitada (depth-limited search)

- Se soluciona el caso de los estados infinitos añadiendo un límite a la profundidad
- Apropiado solo en casos en que se conoce la longitud máxima de la solución de un problema

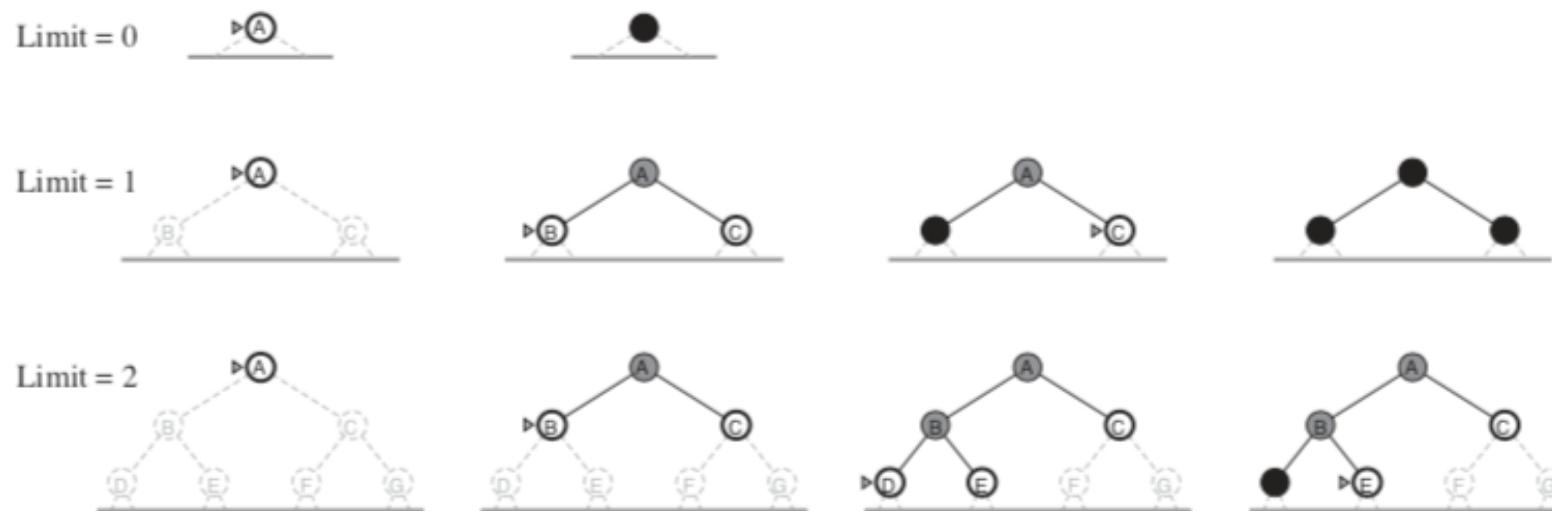
```
function ITERATIVE-DEEPENING-SEARCH(problem) returns a solution node or failure
  for depth = 0 to  $\infty$  do
    result  $\leftarrow$  DEPTH-LIMITED-SEARCH(problem, depth)
    if result  $\neq$  cutoff then return result
```

```
function DEPTH-LIMITED-SEARCH(problem,  $\ell$ ) returns a node or failure or cutoff
  frontier  $\leftarrow$  a LIFO queue (stack) with NODE(problem.INITIAL) as an element
  result  $\leftarrow$  failure
  while not IS-EMPTY(frontier) do
    node  $\leftarrow$  POP(frontier)
    if problem.Is-GOAL(node.STATE) then return node
    if DEPTH(node)  $>$   $\ell$  then
      result  $\leftarrow$  cutoff
    else if not IS-CYCLE(node) do
      for each child in EXPAND(problem, node) do
        add child to frontier
  return result
```

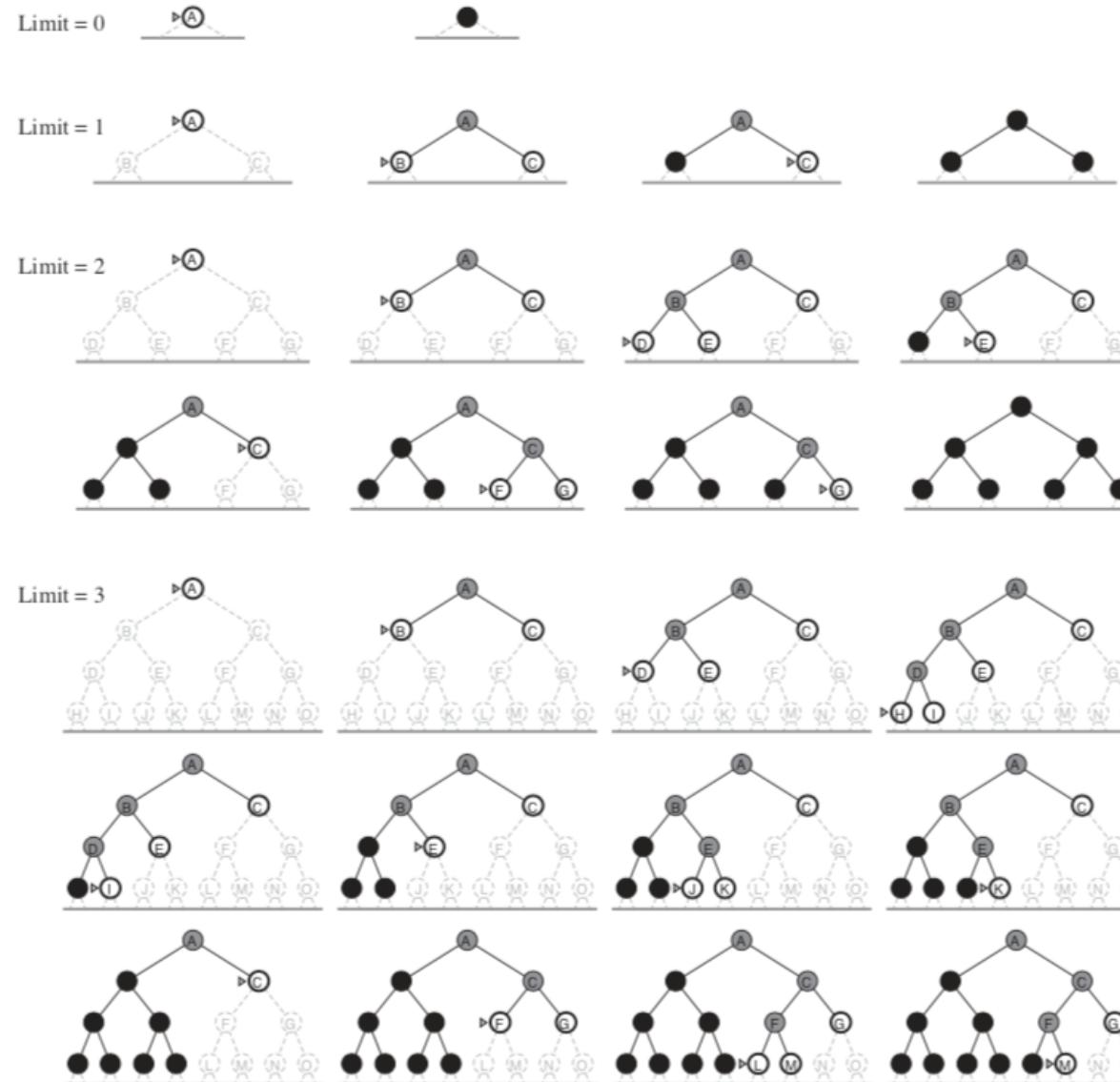
Cutoff: Puede haber solución a más profundidad que l  
Es una búsqueda en árbol no guarda estados alcanzados  
IS-CYCLE no chequea todos los ciclos puede haber bucles

# Búsqueda iterativa que profundiza primero en profundidad (Iterative deepening depth-first search)

```
function ITERATIVE-DEEPENING-SEARCH(problem) returns a solution, or failure
    for depth = 0 to  $\infty$  do
        result  $\leftarrow$  DEPTH-LIMITED-SEARCH(problem, depth)
        if result  $\neq$  cutoff then return result
```

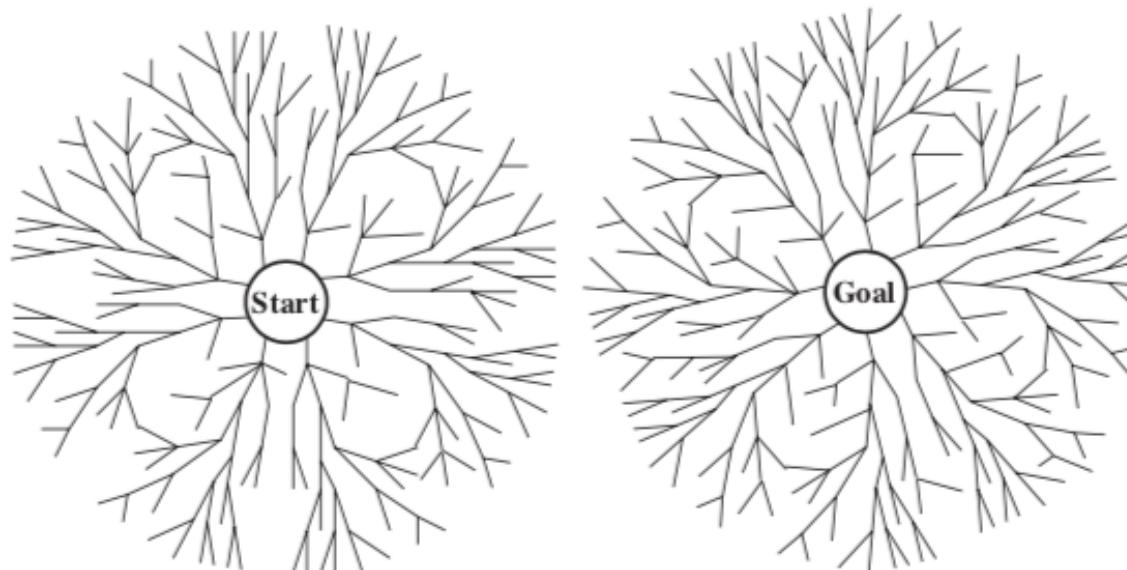


# Búsqueda iterativa que profundiza primero en profundidad (Iterative deepening depth-first search)



## Búsqueda bidireccional

- Dos búsquedas simultáneas:
  - Una desde el estado **inicial**
  - Otra desde el **objetivo**
- Esperando encontrarse en el medio
- Se reemplazan el test de objetivo conseguido con una verificación para ver si las fronteras de las dos búsquedas se cruzan; si lo hacen, se ha encontrado una solución.



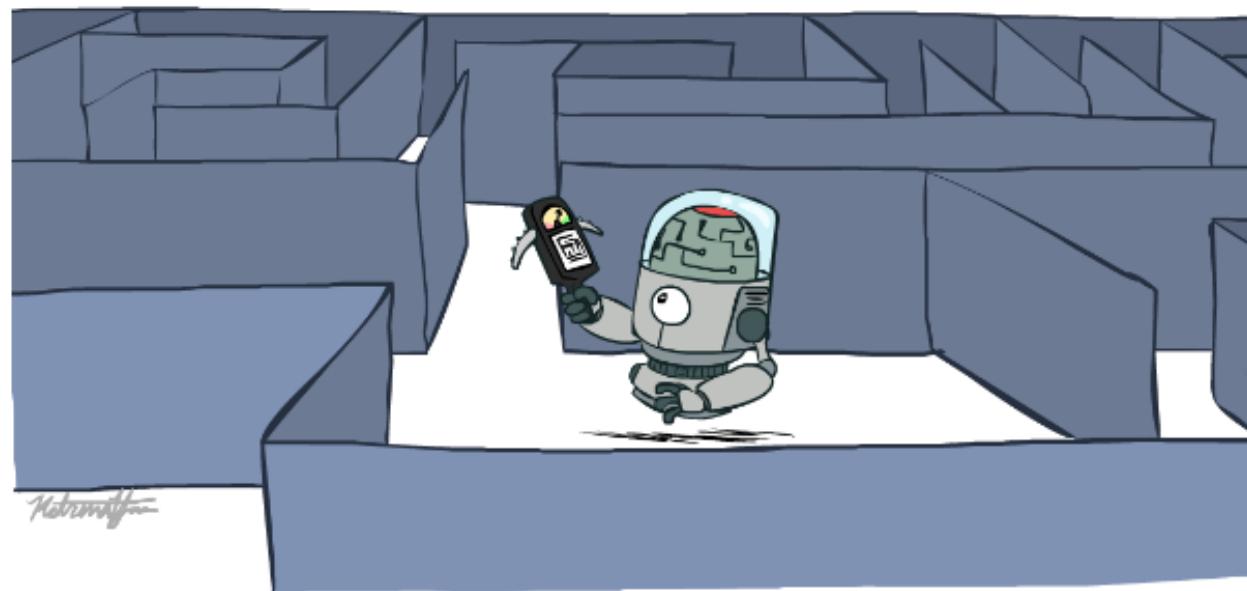
- Su complejidad es menor
- Puede no ser óptimo
- Requiere poder procesar **predecesores**

# Comparación de estrategias de búsqueda no informada

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

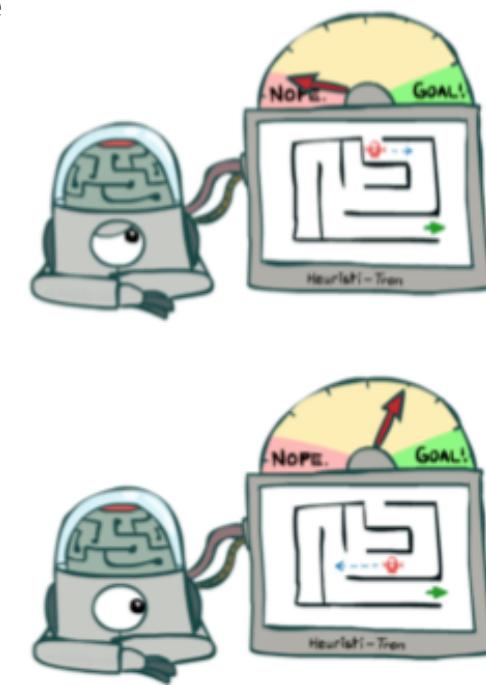
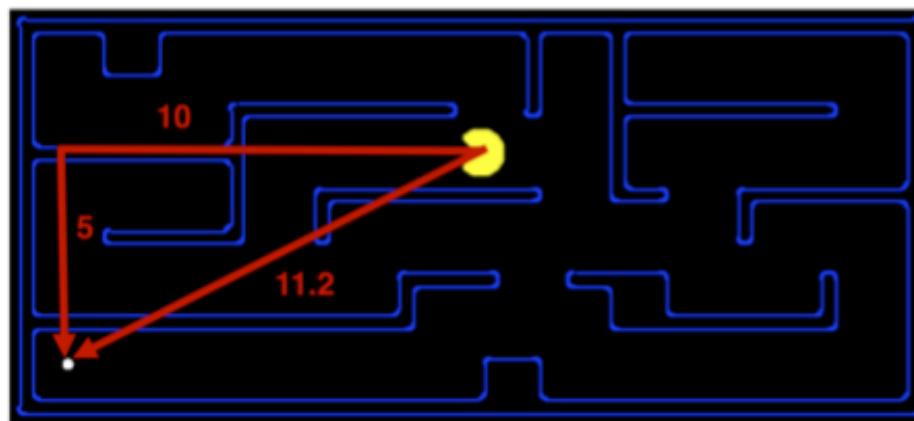
- a completo si b es finito
- b completo si el coste de cada paso  $\geq \epsilon$  para un positivo  $\epsilon$
- c optimo si los costes de cada paso son todos idénticos
- d si ambas direcciones usan breadth-first search.

# Estrategias de búsqueda informadas (heurística)



- Utiliza conocimientos específicos del problema más allá de la definición del problema en sí
- Puede encontrar soluciones de manera más eficiente que una estrategia no informada.
- El enfoque general que consideramos se denomina **búsqueda del mejor primero** (best-first search)
  - Uso de la **función de evaluación  $f(h)$** , que es una estimación de coste. Similar a Uniform-cost-search, pero usando  $f$  en lugar de  $g$
- La elección de  $f$  determina la estrategia de búsqueda. La mayoría de los algoritmos de búsqueda del mejor primero incluyen como componente de  $f$  una **función heurística**, denotada  $h(n)$ :

$h(n)$  = coste estimado de la ruta más barata desde el estado en el nodo  $n$  hasta un estado objetivo.

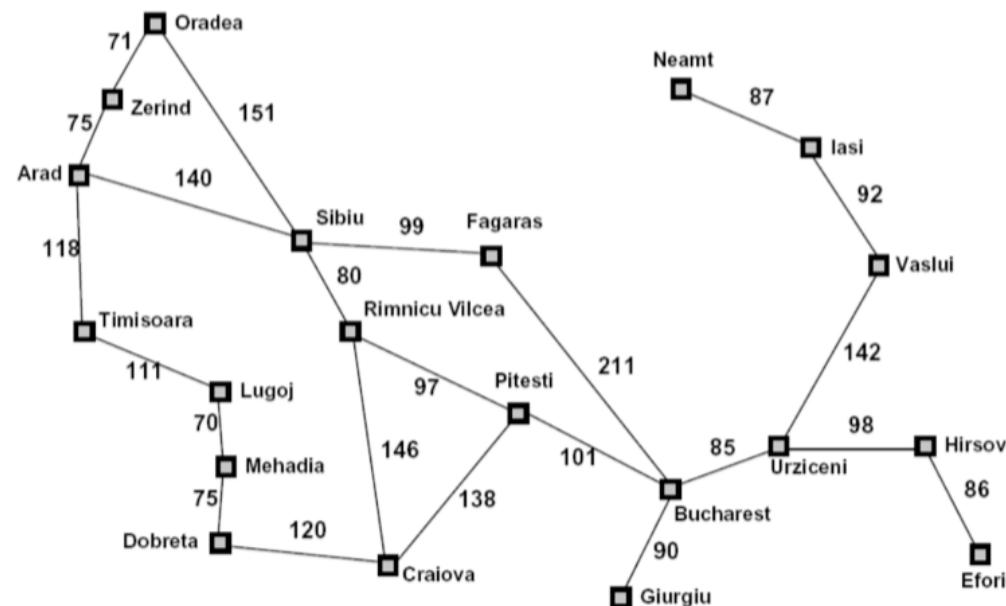


# Búsqueda codiciosa del mejor primero (Greedy best-first search)

- Intenta expandir el nodo más cercano al objetivo
- Evalúa los nodos utilizando solo la función heurística; es decir,  $f(n) = h(n)$ .



- En el caso del ejemplo de Rumanía, usamos la distancia en linea recta  $h_{SLD}$
- La función heurística no se puede obtener de la descripción del problema



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

$h(x)$

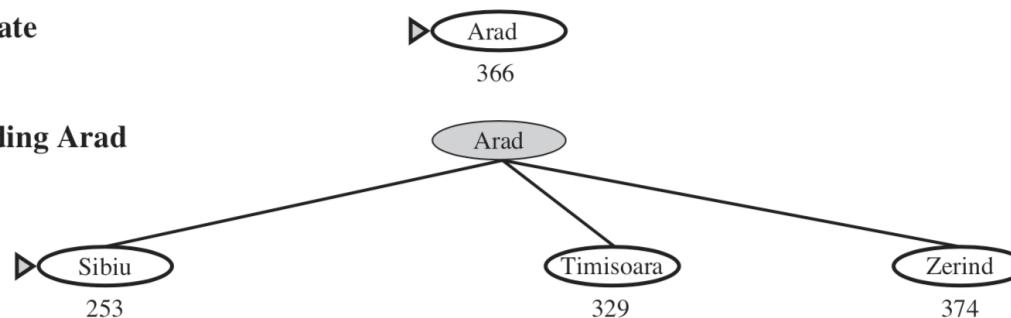
# Búsqueda codiciosa del mejor primero (Greedy best-first search)

(ir a lo más cercano)

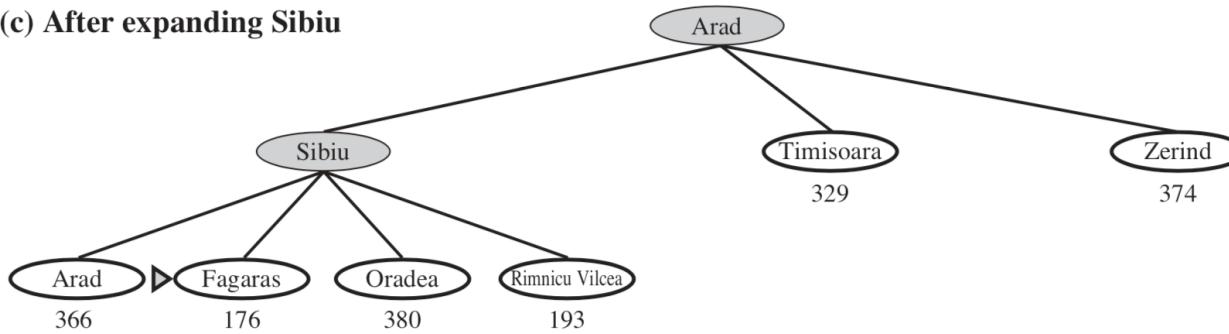
(a) The initial state



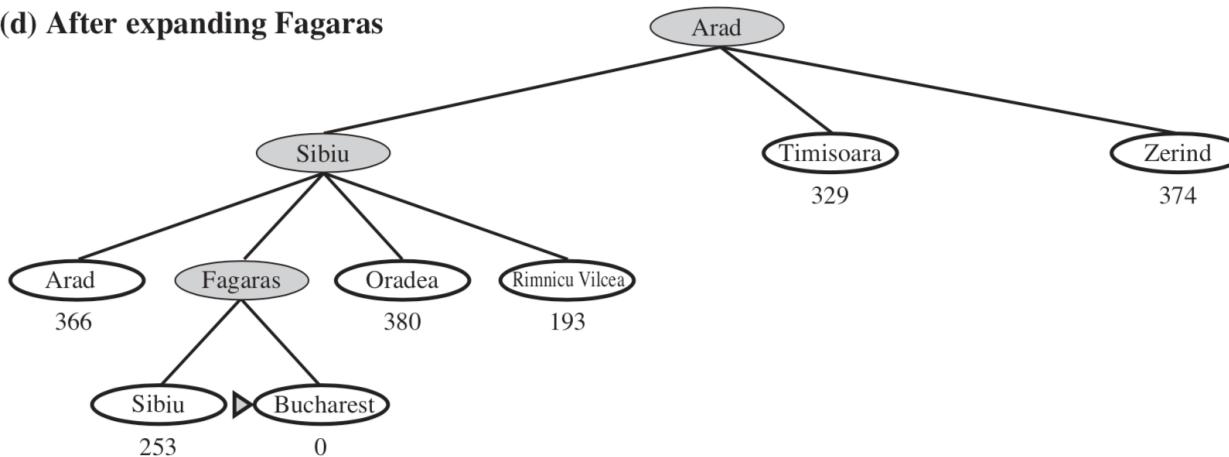
(b) After expanding Arad



(c) After expanding Sibiu

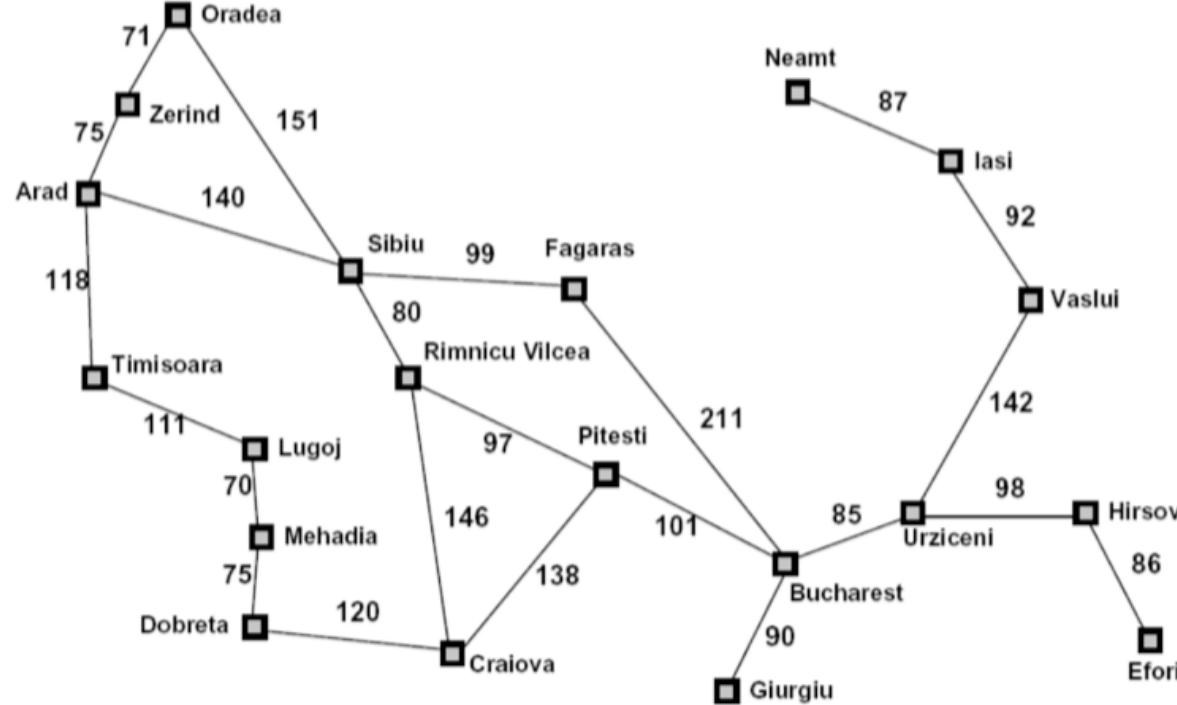


(d) After expanding Fagaras



# Búsqueda codiciosa del mejor primero (Greedy best-first search)

- No es óptimo: Rimnicu Vilcea y Pitesti es más corto
- No es completo: Ej: Iasi a Fagaras



Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobrete	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

$h(x)$

## Búsqueda A\* (A\* Search)

El coste donde estamos + estimación hasta donde queremos llegar.

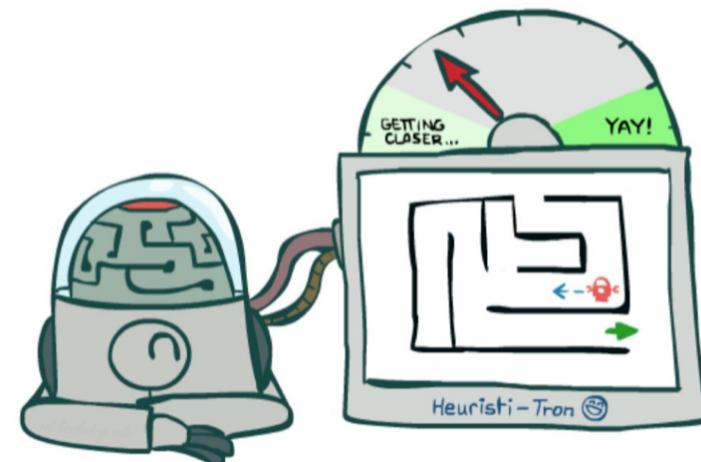
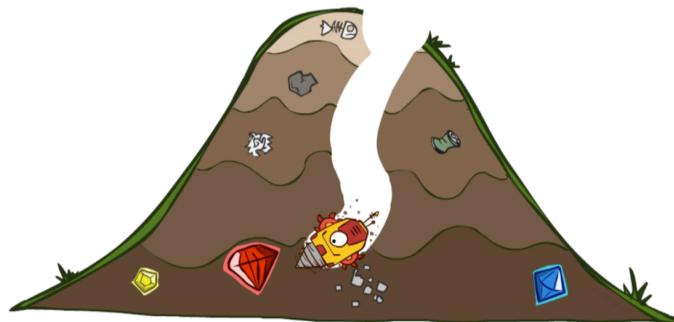
- La forma más conocida de best-first es A\*
- Combina el coste de llegar al nodo  $g(n)$  con el coste estimado hasta el destino  $h(n)$

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

- $f(n)$  es el coste estimado de la solución más barata a través de  $n$
- A\* requiere:
  - usar una **heurística admisible** — aquella que no sobreestime  $g(n)$
  - Se requiere **consistencia** (aka **monotonidad**). Para un nodo  $n$  y todos los sucesores  $n'$  de  $n$  generados por cualquier acción  $a$

que sea decreciente en el tiempo

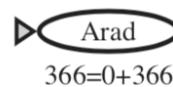
$$h(n) \leq c(n, a, n') + h(n')$$



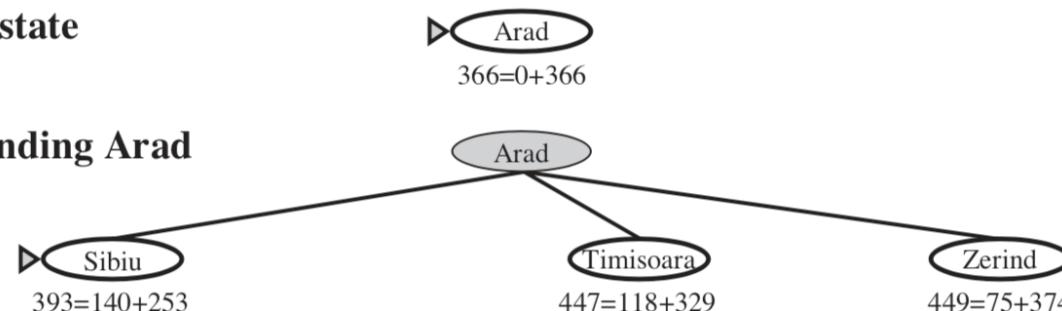
# Búsqueda A\*

## (A\* Search)

**(a) The initial state**



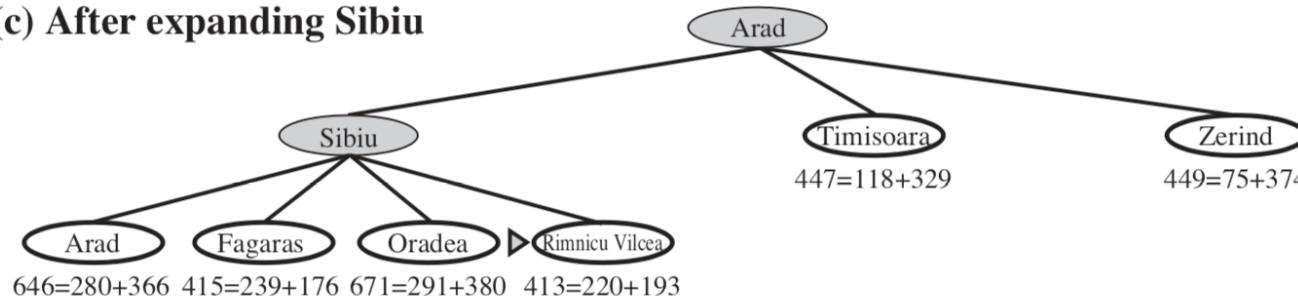
**(b) After expanding Arad**



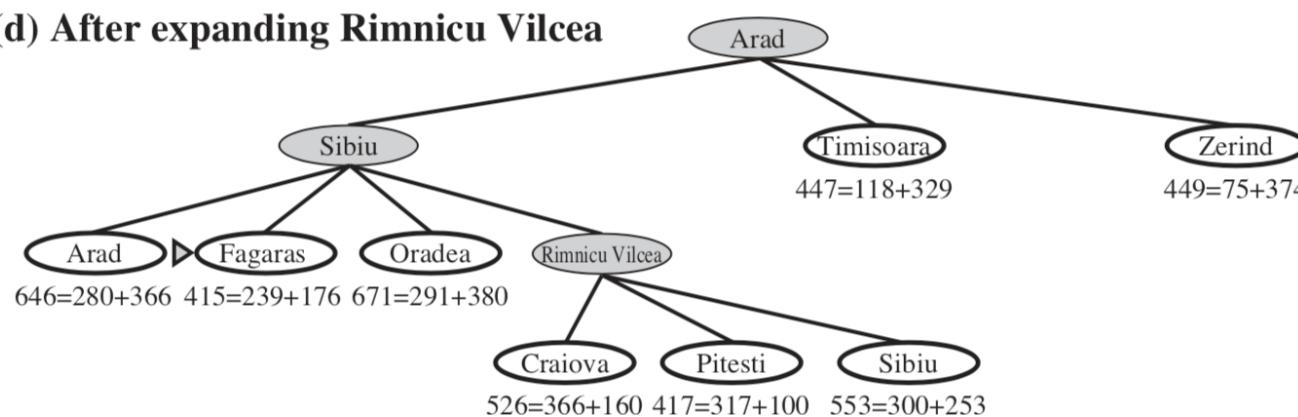
Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

$h(x)$

**(c) After expanding Sibiu**

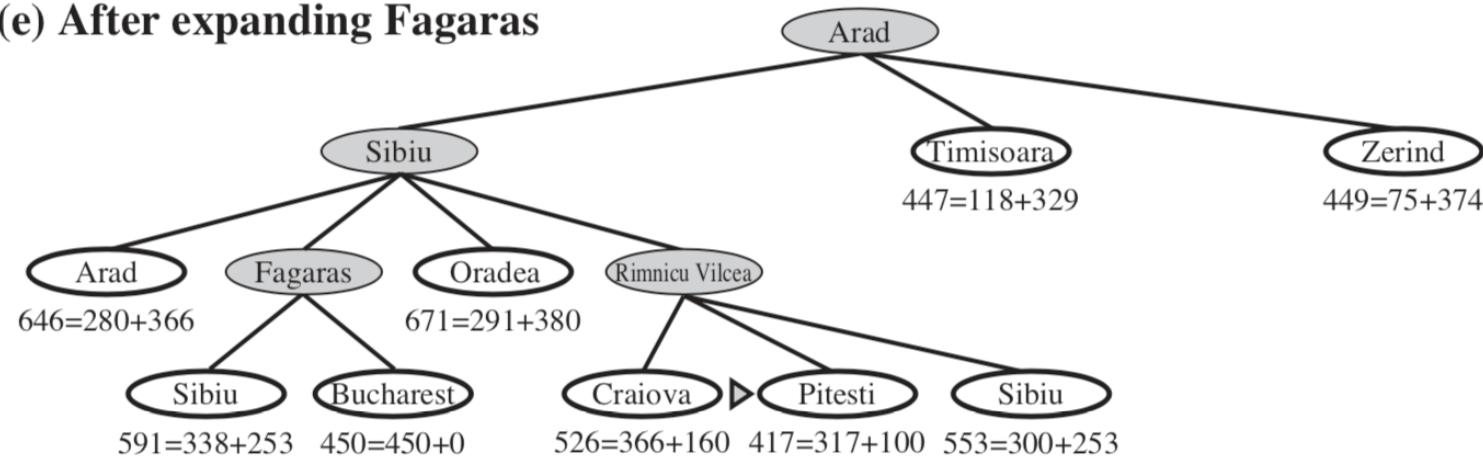


**(d) After expanding Rimnicu Vilcea**



## Búsqueda A\* (A\* Search)

(e) After expanding Fagaras

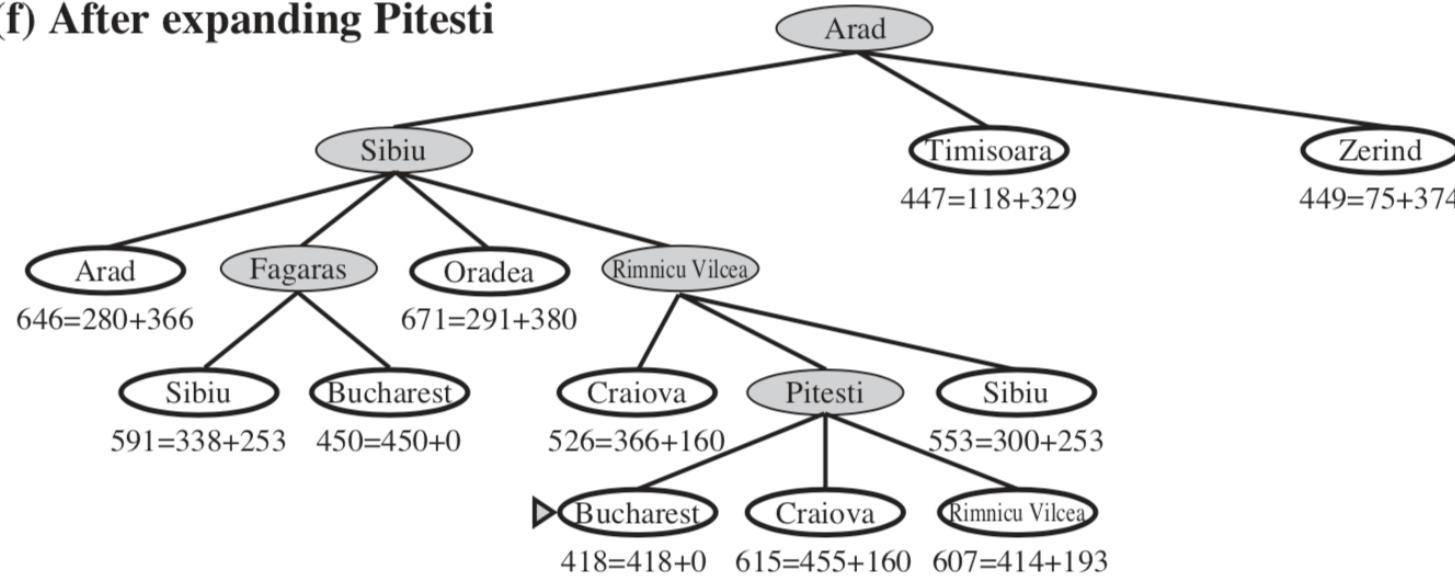


Straight-line distance to Bucharest

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

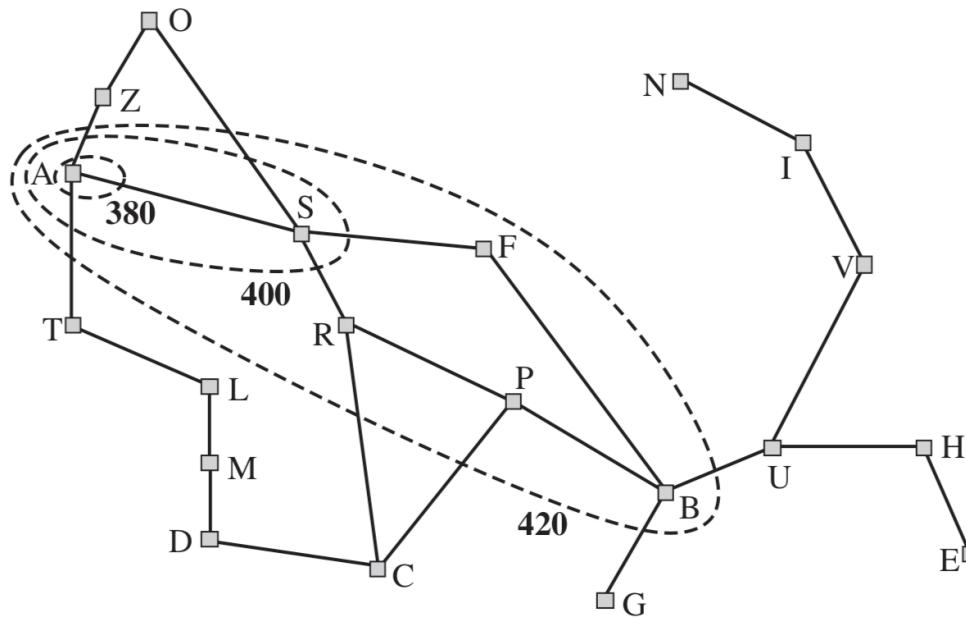
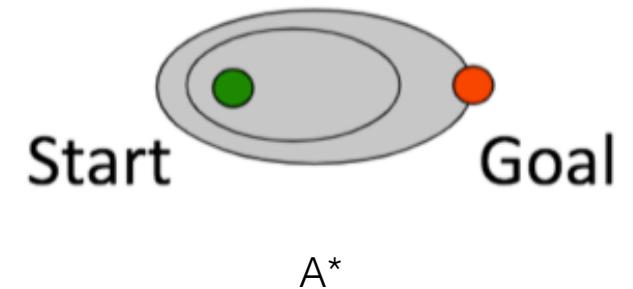
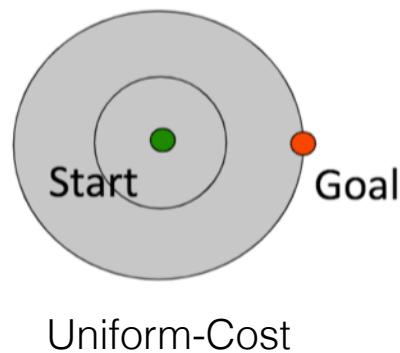
$h(x)$

(f) After expanding Pitesti



## Búsqueda A\*

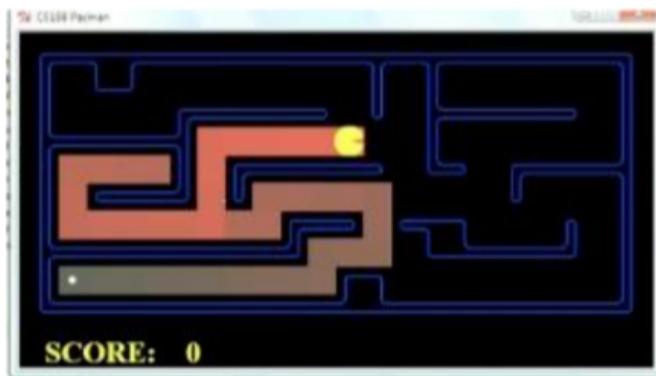
### (A\* Search)



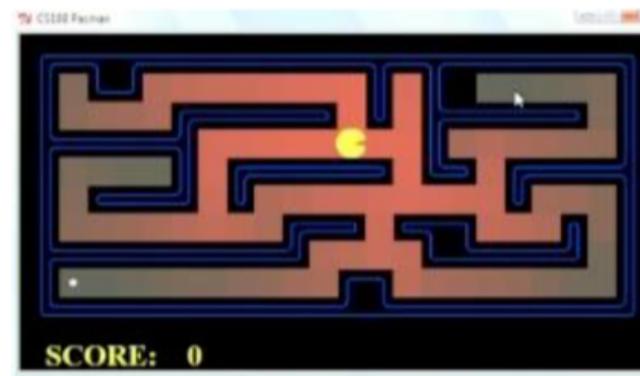
Contorno de las soluciones  
con  $f = 380, 400$  y  $420$

## Búsqueda A\*

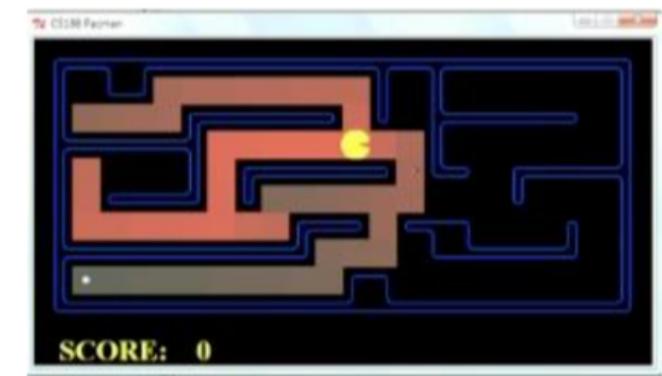
### (A\* Search)



Greedy



Uniform Cost

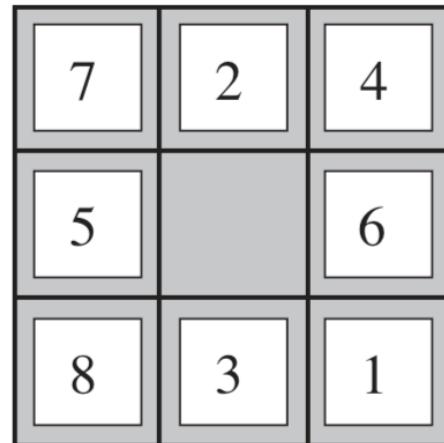


A\*

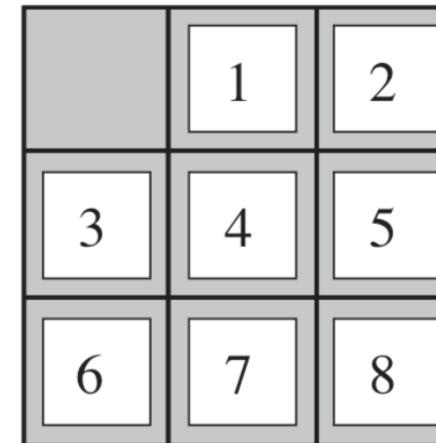
Una búsqueda A\* es completa, óptima y ópticamente eficiente con comparación con el resto de los algoritmos

## Funciones heurísticas

- El puzzle de 8 fichas es uno de los primeros problemas de búsqueda heurística
- heurísticas:
  - $h_1$ : número de fichas mal colocadas:  $h(\text{inicio}) = 8$
  - $h_2$ : la suma de las distancias de las fichas hasta su posición correcta:  $h(\text{inicio}) = 18$
- ¿Son admisibles? La solución más corta del ejemplo conlleva 26 movimientos



Start State



Goal State

## Ejercicios

3.3

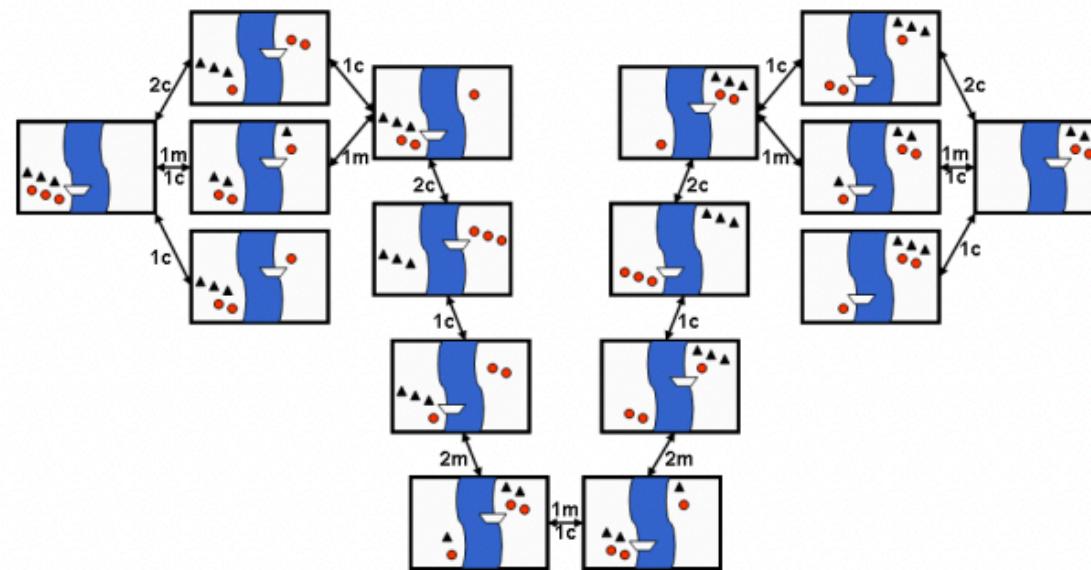
1. Tu objetivo es sacar a un robot de un laberinto. El robot comienza en el centro del laberinto mirando al norte. Puede girar el robot para que mire al norte, este, sur u oeste. Puedes dirigir al robot para que avance una cierta distancia, aunque se detendrá antes de chocar contra una pared.
  - a. Formula este problema. ¿Cómo de grande es el espacio de estados?
  - b. Al navegar por un laberinto, el único lugar en el que necesitamos girar es en la intersección de dos o más pasillos. Reformula este problema utilizando esta observación. ¿Cómo de grande es el espacio de estados ahora?
  - c. Desde cada punto del laberinto, podemos movernos en cualquiera de las cuatro direcciones hasta llegar a un punto de giro, y esta es la única acción que debemos realizar. Reformula el problema utilizando estas acciones. ¿Necesitamos tener en cuenta la orientación del robot ahora?
  - d. En nuestra descripción inicial del problema ya nos abstrajimos del mundo real, restringiendo acciones y eliminando detalles. Enumere tres de esas simplificaciones que hicimos.

3.9

## Ejercicios

2. El **problema de los misioneros y caníbales** se suele plantear de la siguiente manera. Tres misioneros y tres caníbales están a un lado de un río, junto con un bote que puede llevar a una o dos personas. Encuentra una manera de llevar a todos al otro lado sin dejar un grupo de misioneros en un lugar superado en número por los caníbales en ese lugar. Este problema es famoso en la IA porque fue el tema del primer artículo que abordó la formulación del problema desde un punto de vista analítico (Amarel, 1968).

- Formula el problema con precisión, haciendo solo las distinciones necesarias para asegurar una solución válida. Dibuja un diagrama del espacio de estados completo.
- Implementa y resuelve el problema de manera óptima utilizando un algoritmo de búsqueda apropiado. ¿Es una buena idea comprobar si hay estados repetidos?
- ¿Por qué crees que a la gente le cuesta resolver este rompecabezas, dado que el espacio de estados es tan simple?



## Ejercicios

3.23

3. Traza la operación de búsqueda A\* aplicada al problema de llegar a Bucarest desde Lugoj utilizando la heurística de distancia en línea recta. Es decir, muestre la secuencia de nodos que considerará el algoritmo y la puntuación  $f$ ,  $g$  y  $h$  para cada nodo.

3.25  
4. El algoritmo de ruta heurística (Pohl, 1977) es una búsqueda del mejor primero en la que la función de evaluación es  $f(n) = (2 - w) g(n) + wh(n)$ . ¿Para qué valores de  $w$  es esto completo? ¿Para qué valores es óptimo, asumiendo que  $h$  es admisible? ¿Qué tipo de búsqueda realiza esto para  $w = 0$ ,  $w = 1$  y  $w = 2$ ?

## Ejercicios

3.26

4. Considera la versión ilimitada de la cuadrícula 2D regular de la Figura mostrada. El estado inicial está en el origen,  $(0,0)$  y el estado objetivo está en  $(x, y)$ .
- ¿Cuál es el factor de ramificación  $b$  en este espacio de estados?
  - ¿Cuántos estados distintos hay a la profundidad  $k$  (para  $k > 0$ )?
  - ¿Cuál es el número máximo de nodos expandidos por la búsqueda del árbol primero en amplitud?
  - ¿Cuál es el número máximo de nodos expandidos por la búsqueda de gráfico de amplitud primero?
  - ¿Es  $h = |u - x| + |v - y|$  una heurística admisible para un estado en  $(u, v)$ ? Explicar.
  - ¿Cuántos nodos se expanden mediante la búsqueda de gráficos A\* usando  $h$ ?
  - ¿ $h$  sigue siendo admisible si se eliminan algunos enlaces?
  - ¿Sigue siendo admisible  $h$  si se agregan algunos vínculos entre estados no adyacentes?

