UNIVERSIDAD EAFIT
VIGILADA | MINEDUCACIÓN

Escuela de
Ciencias Aplicadas
e Ingeniería

Artificial Intelligence

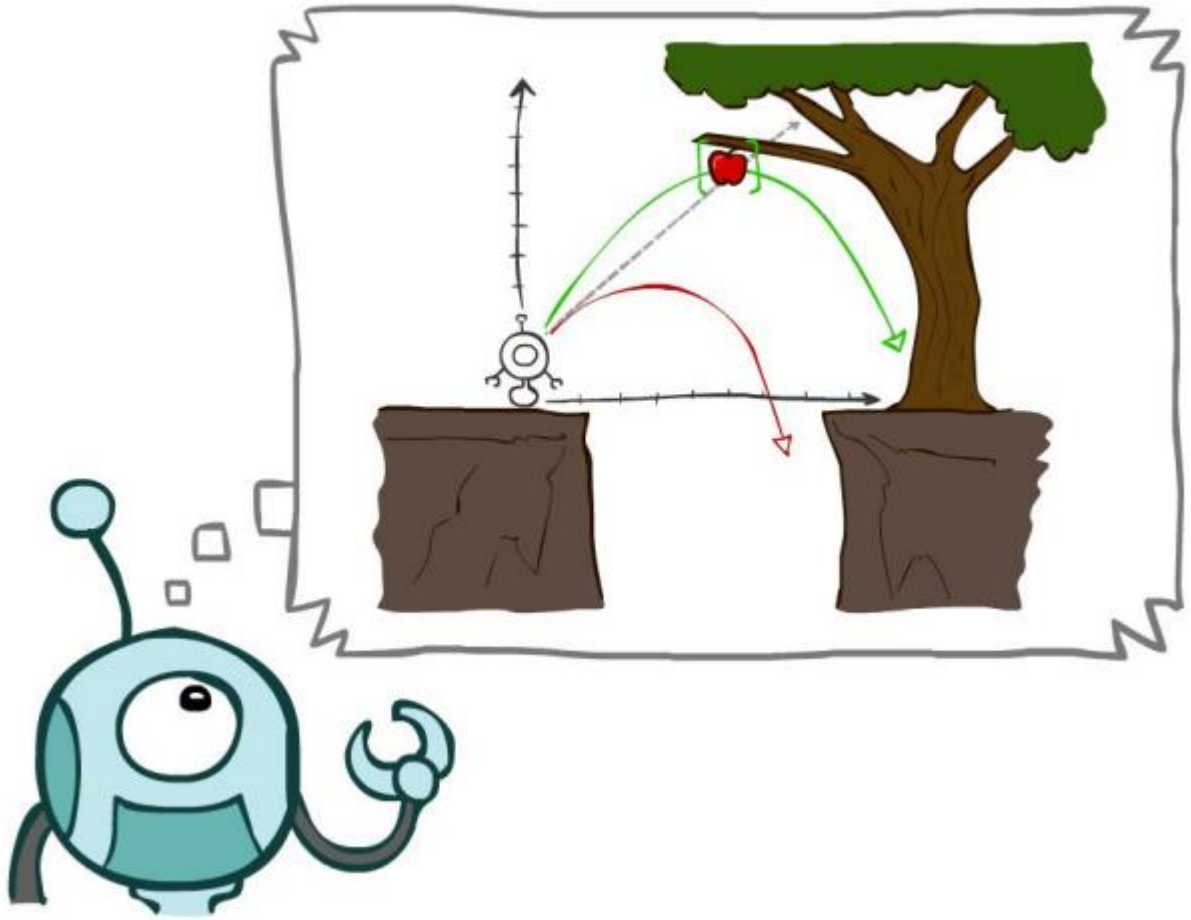
Lecture02 – Problem Formulation and Uninformed Search

Contenido

UNIVERSIDAD EAFIT
VIGILADA | MINEDUCACIÓN
Escuela de Ciencias Aplicadas e Ingeniería

1. Agentes que planifican
2. Problemas de Búsqueda
3. Arboles de búsqueda (Search Trees)
4. Estrategias de Búsqueda Ciega (Uninformed Search)

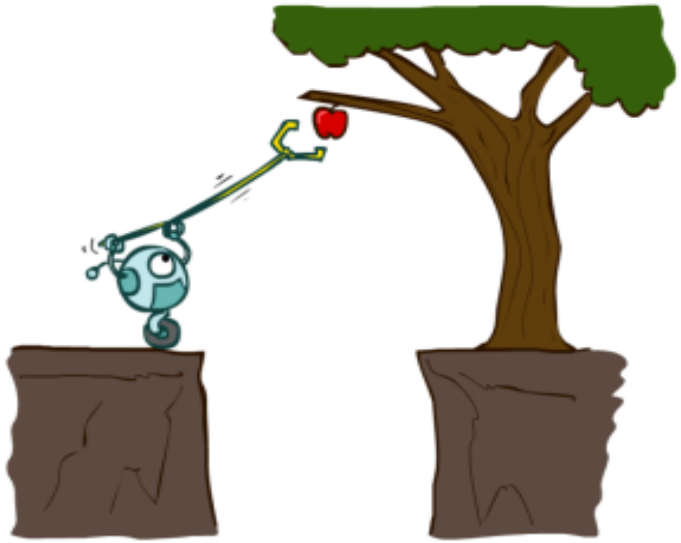
Agentes que planifican

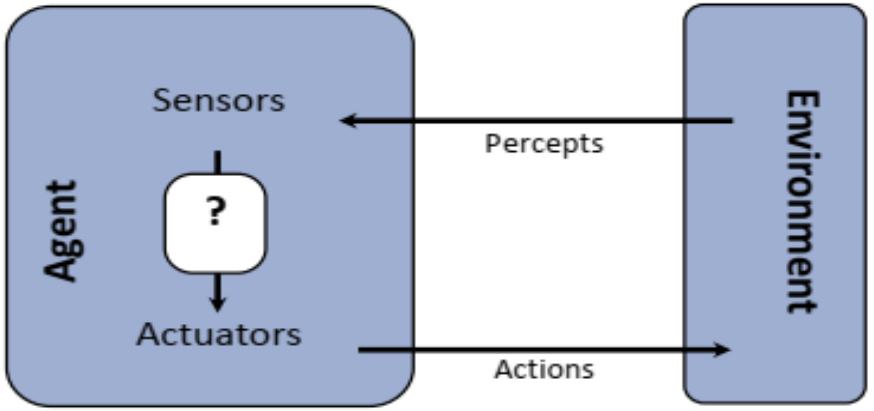


Agentes que planifican

Cuando la acción correcta a tomar no es inmediatamente obvia, un agente puede necesitar planificar con anticipación:

A tal agente se le llama **Agente que planifica con antelación** y el proceso computacional que lleva a cabo se llama **búsqueda.**





Agentes que planifican

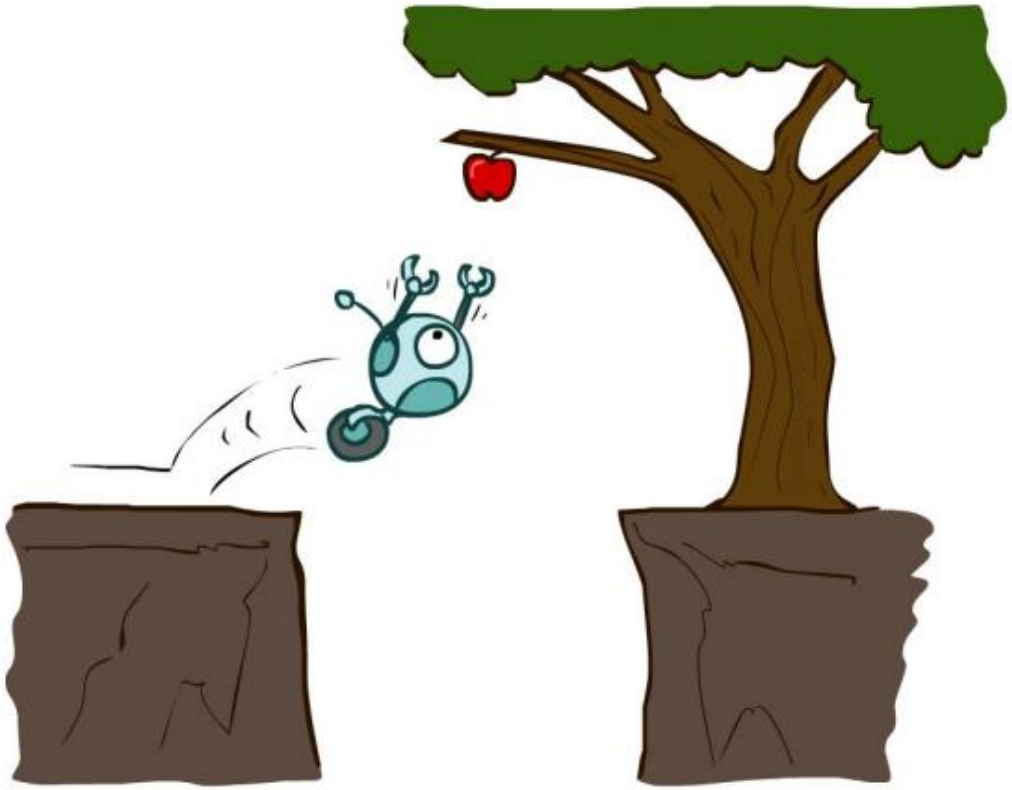
¿Por qué nos importa la búsqueda?

- Problemas de trazado de enrutamiento (tanto para personas como para robots)
- Diseño de chips (enrutamiento de cables)
- Problemas de planificación de recursos
- Diseño de proteínas
- Análisis del lenguaje
- Traducción automática
- Videojuegos
- Reconocimiento de voz

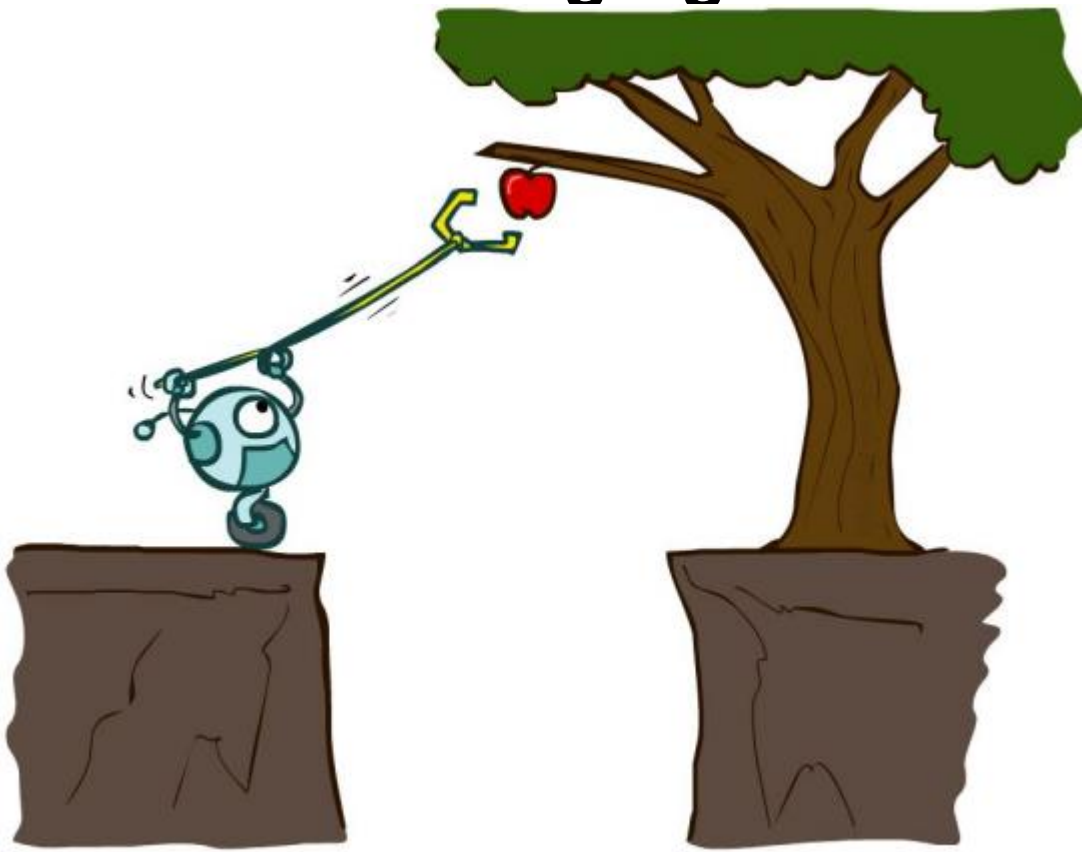
Agentes que planifican

UNIVERSIDAD EAFIT
Escuela de Ciencias Aplicadas e Ingeniería

Reflex Agents



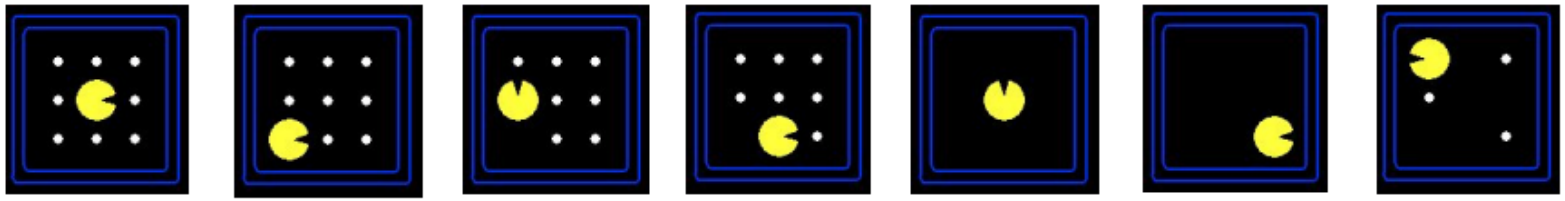
Planning Agents



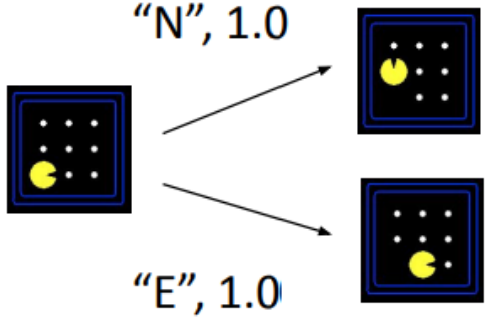
Problemas de búsqueda

Un problema de búsqueda consta de:

Un espacio de estados
(State Space)



Una función sucesora
(Actions and costs)



Un estado inicial (Start State) y una prueba de objetivo (Goal Test).

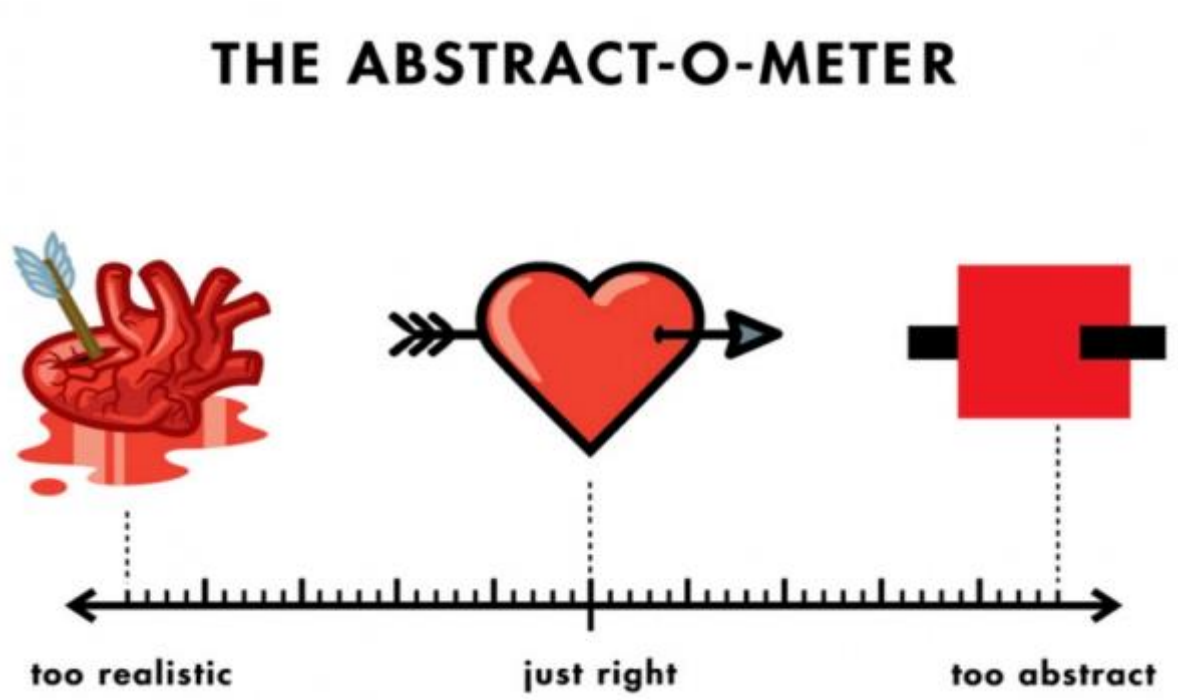
Una solución es una secuencia de acciones (un plan) que transforma el estado inicial en un estado objetivo (Goal State).

Problemas de búsqueda

LOS PROBLEMAS DE BÚSQUEDA SON MODELOS

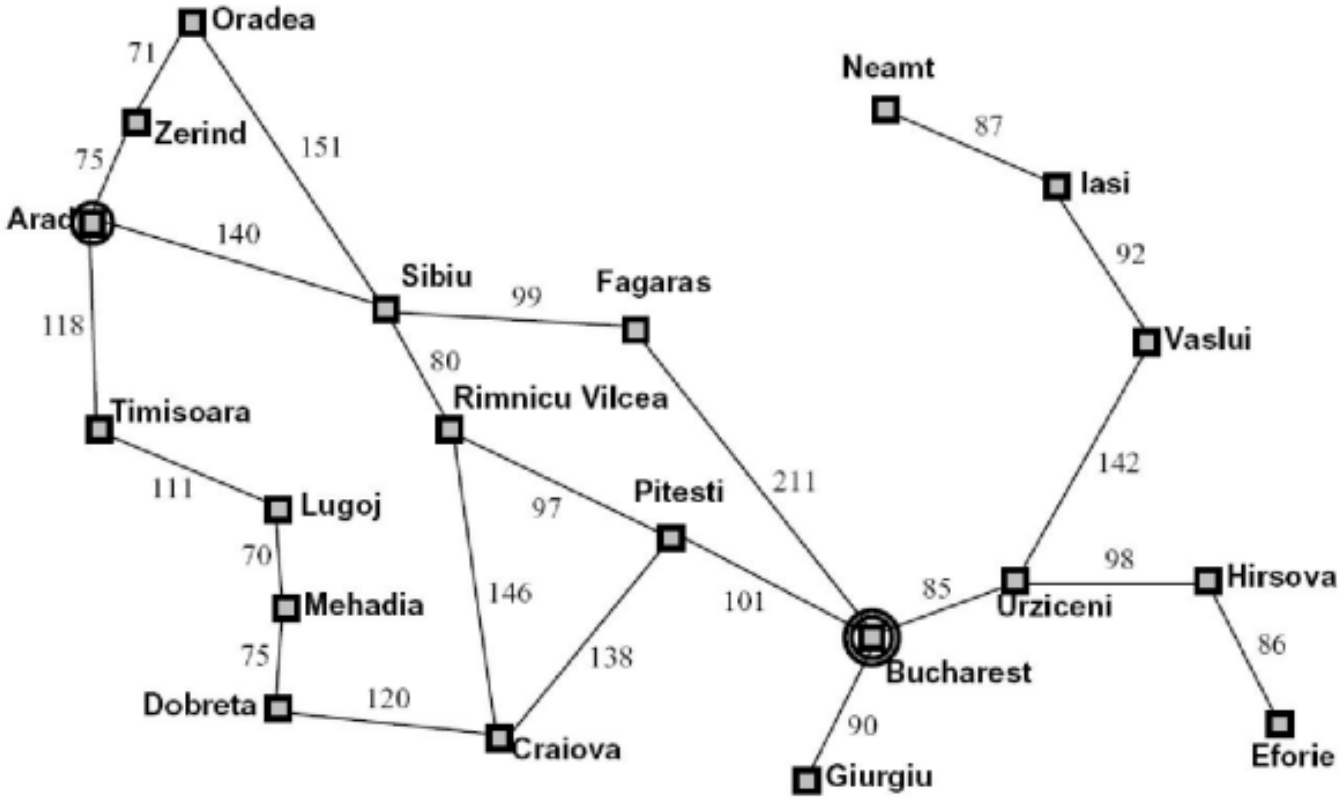
Problemas de búsqueda

Para iniciar con el estudio de algoritmos de búsqueda debemos estudiar la forma adecuada de formular problemas…la clave es… Abstracción

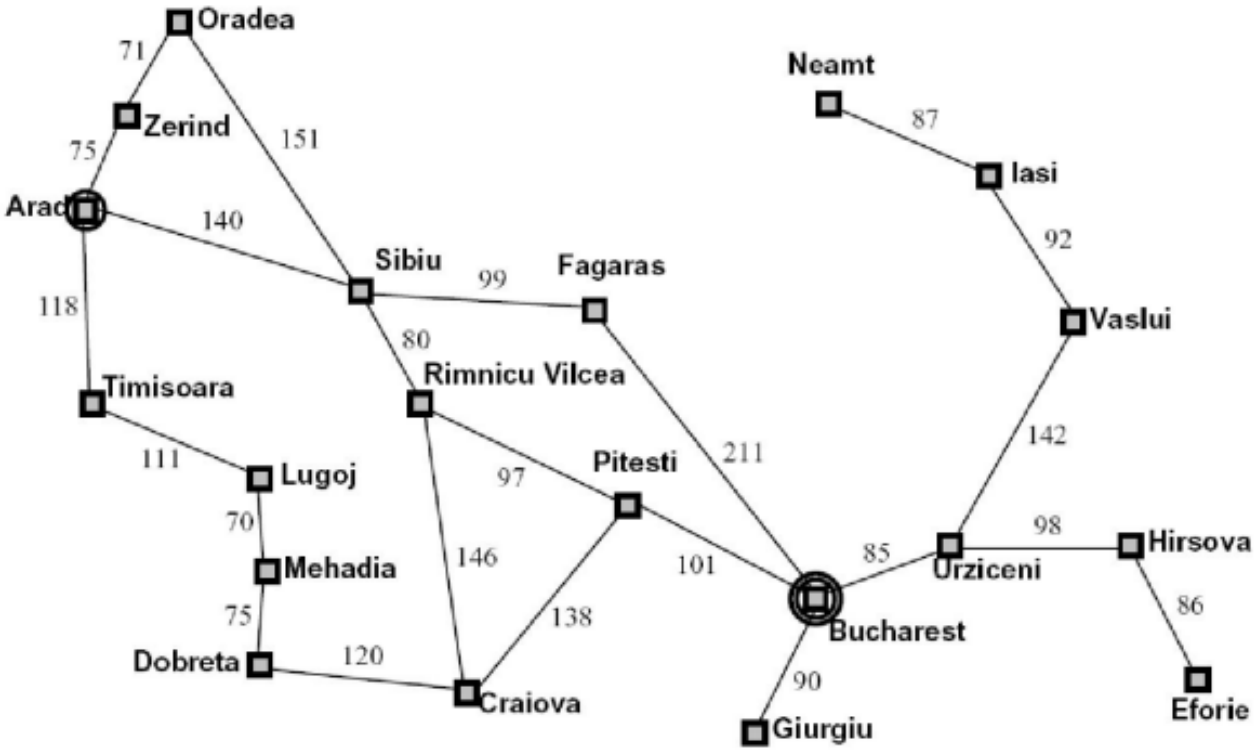


Problemas de búsqueda

Un agente se encuentra en Arad y necesita llegar a Bucarest; sin conocimiento previo del entorno, no puede determinar cuál de los caminos disponibles conduce de manera más eficiente al objetivo.



Problemas de búsqueda



- State space:
  - Cities
- Successor function:
  - Roads: Go to adjacent city with cost = distance
- Start state:
  - Arad
- Goal test:
  - Is state == Bucharest?
- Solution?

Problemas de búsqueda

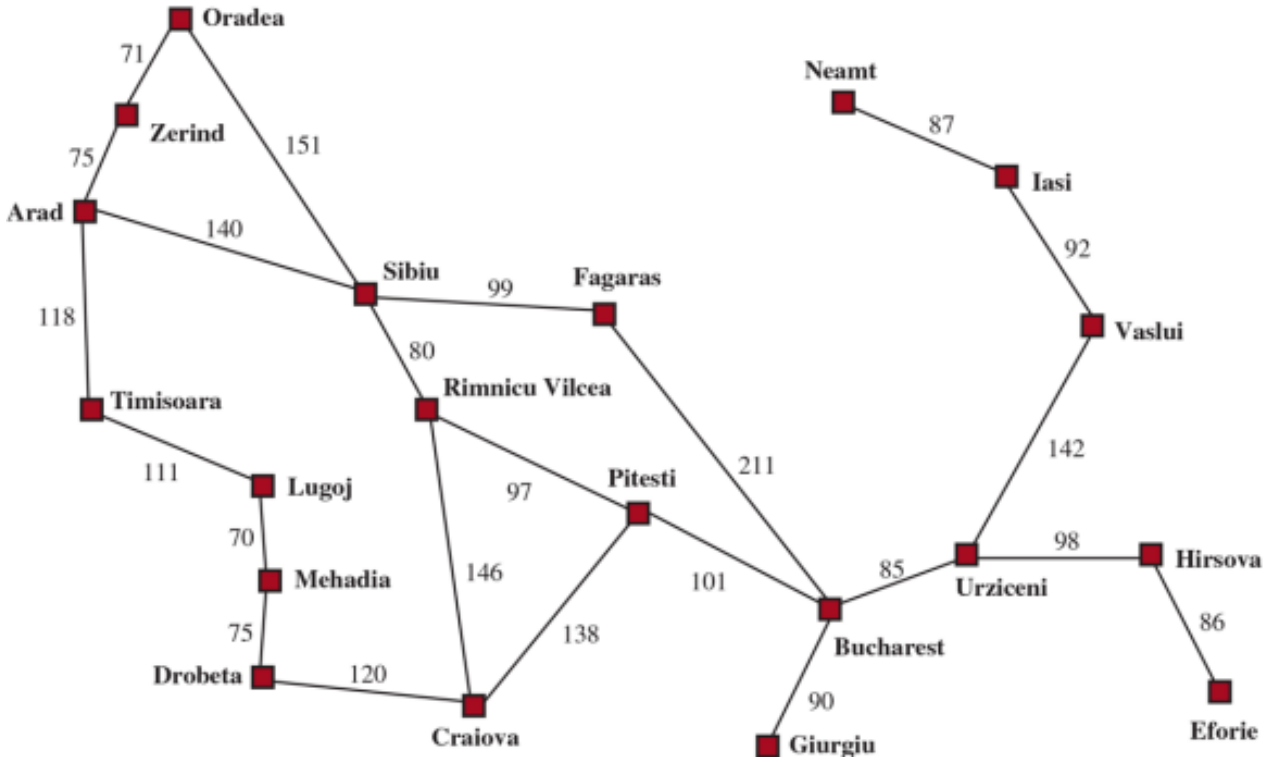
UNIVERSIDAD EAFIT
Escuela de Ciencias Aplicadas e Ingeniería

- **PASO 1: Definición del objetivo**

El agente adopta el objetivo de llegar a Bucarest. Los objetivos organizan las acciones a considerar.

- **PASO 2: Formulación**

El agente elabora una descripción de los estados y acciones necesarios para alcanzar el objetivo.



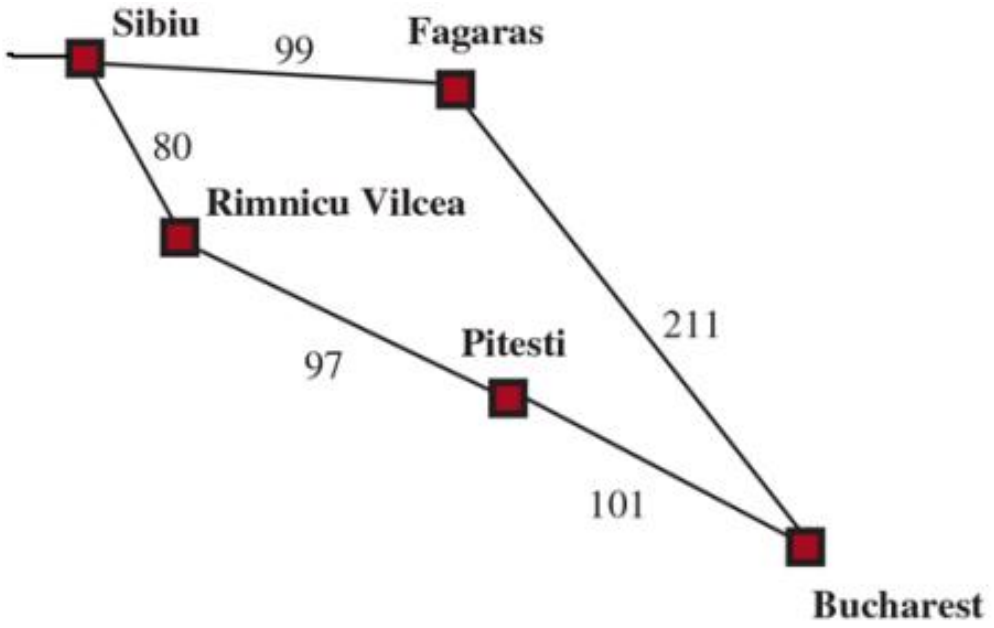
Problemas de búsqueda

- **PASO 3 BÚSQUEDA:**

El agente simula secuencias de acciones en su modelo, buscando hasta encontrar una secuencia de acciones que alcance el objetivo. A tal secuencia se le llama solución

- **PASO 4: EJECUCIÓN**

Si el modelo es correcto, una vez que el agente ha encontrado una solución, puede ejecutar la secuencia de acciones



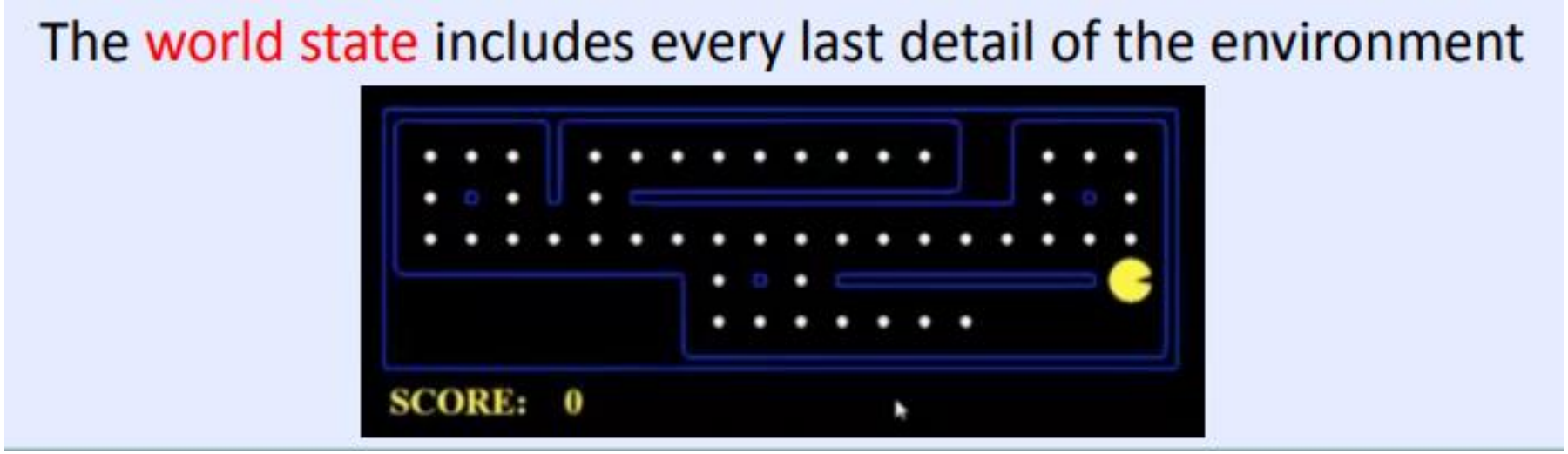
Problemas de búsqueda

UNIVERSIDAD EAFIT
Escuela de Ciencias Aplicadas e Ingeniería

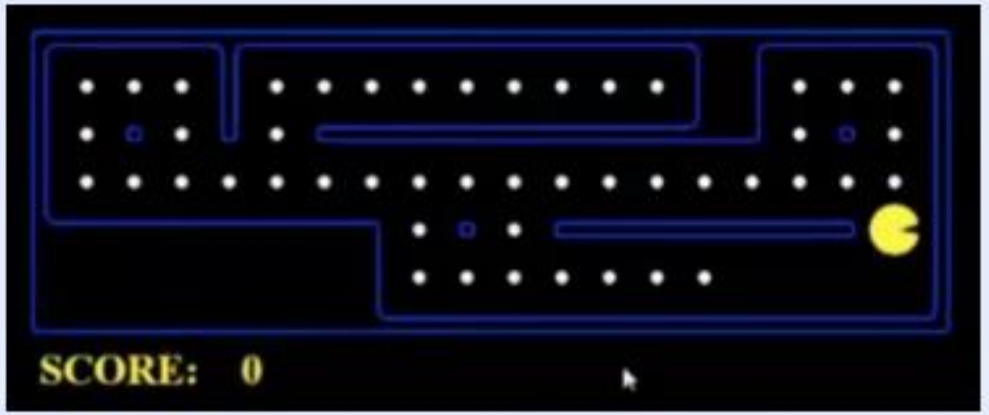
Entonces… Como podemos empezar a definir un problema?

Problemas de búsqueda

Define el problema y el **State Space**: Un conjunto de posibles **estados** en los que el entorno puede estar.



Problemas de búsqueda



A search state keeps only the details needed for planning (abstraction)

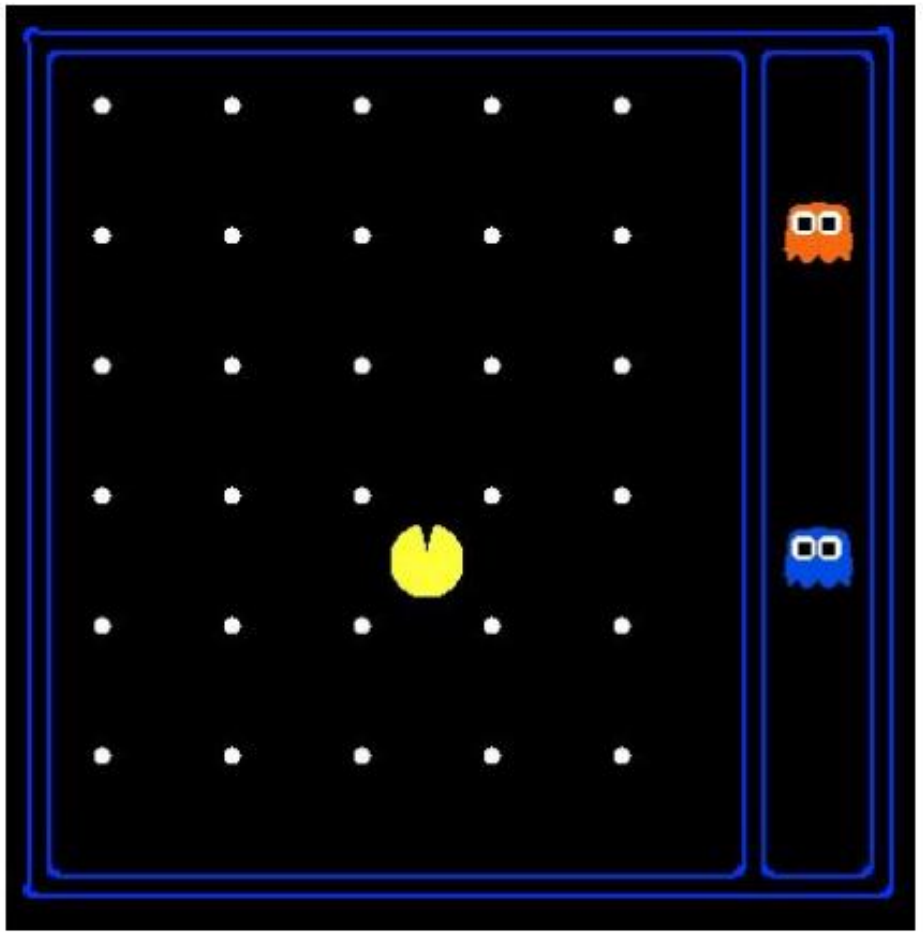
- Problem: Pathing
  - States: (x,y) location
  - Actions: NSEW
  - Successor: update location only
  - Goal test: is (x,y)=END

- Problem: Eat-All-Dots
  - States: {(x,y), dot booleans}
  - Actions: NSEW
  - Successor: update location and possibly a dot boolean
  - Goal test: dots all false

Problemas de búsqueda

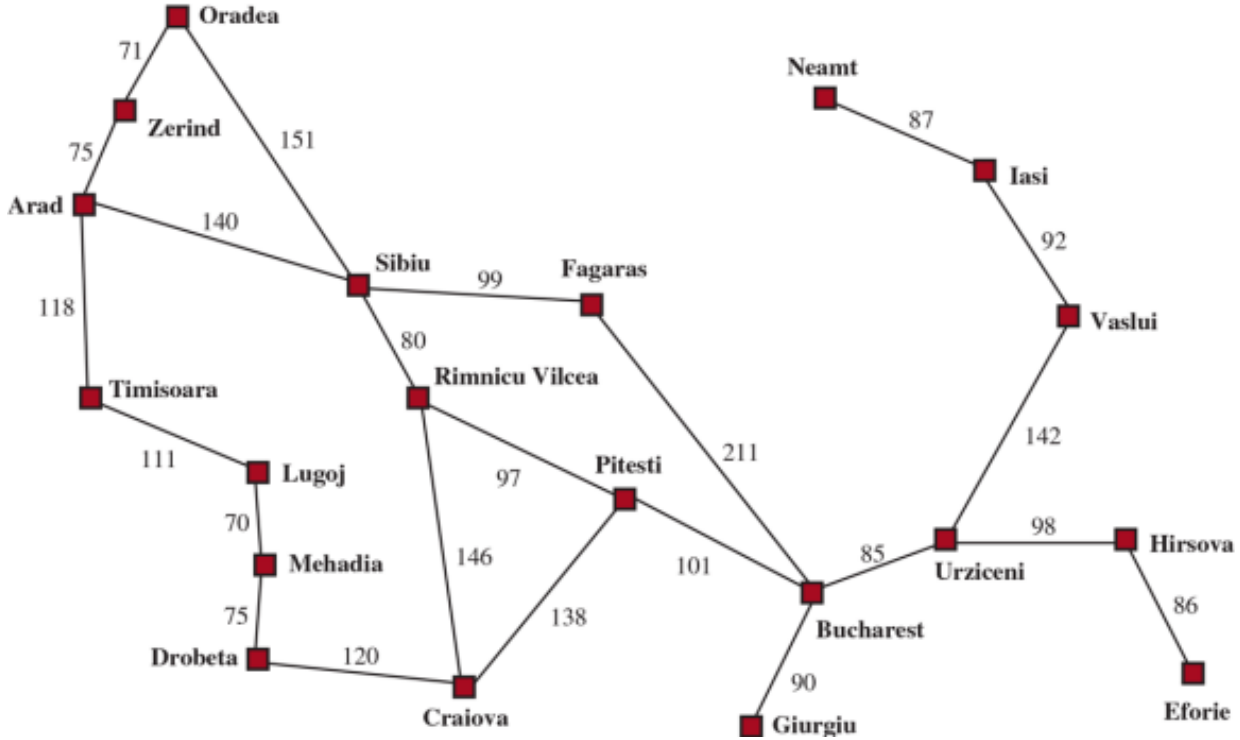
- World state:
  - Agent positions: 120
  - Food count: 30
  - Ghost positions: 12
  - Agent facing: NSEW

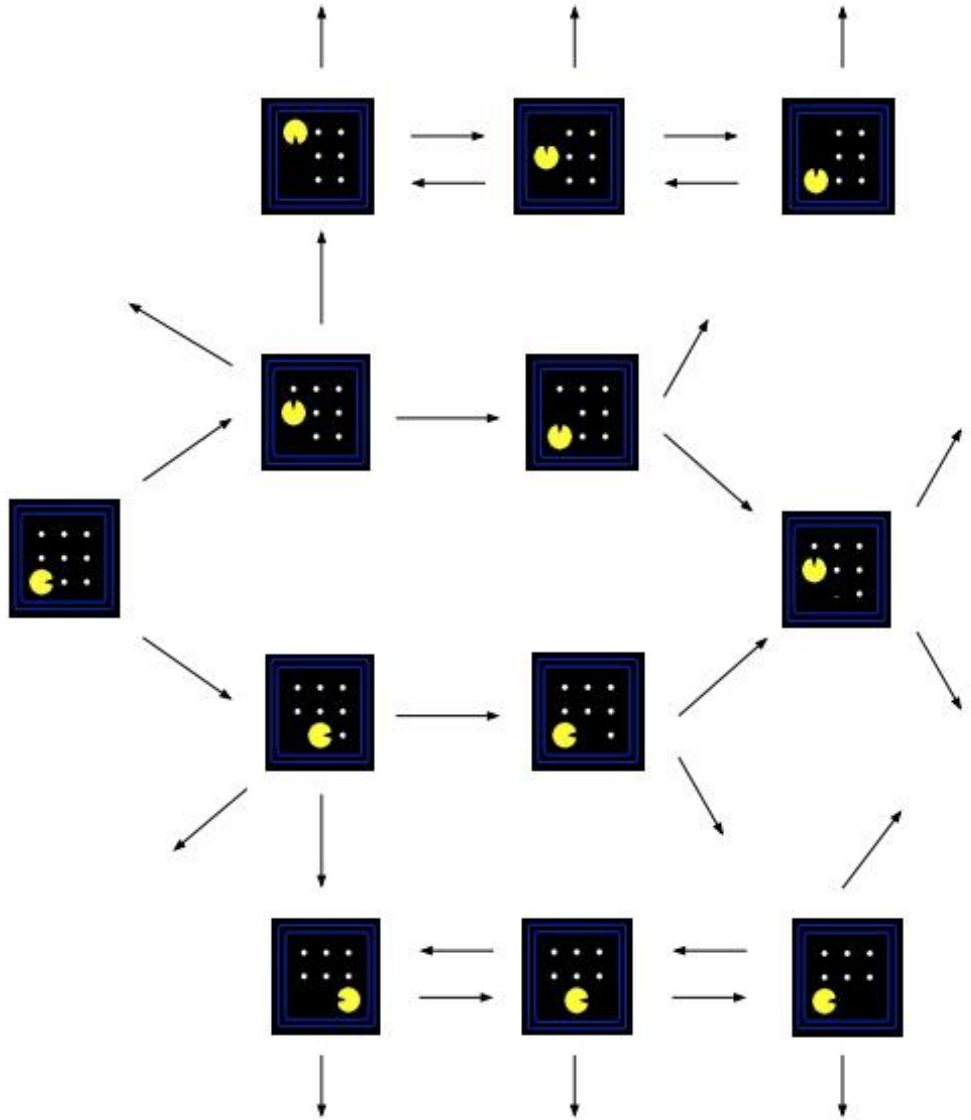
- How many
  - World states?
    $120x(2^{30})x(12^2)x4$
  - States for pathing?
    120
  - States for eat-all-dots?
    $120x(2^{30})$



PROBLEMAS DE BUSQUEDA

El espacio de estados puede representarse como un grafo en el que los vértices son estados y los arcos dirigidos entre ellos son acciones.





PROBLEMAS DE BUSQUEDA

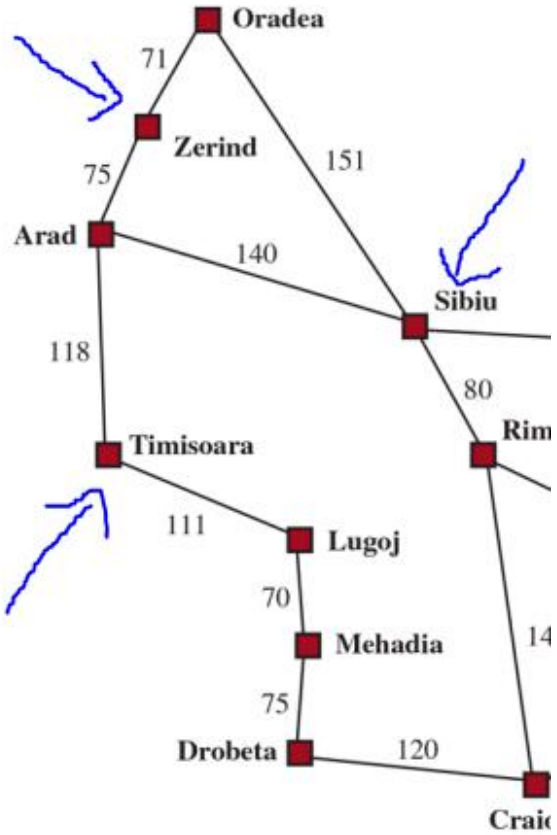
COMPOSICIÓN:

1. ACTIONS:

Dado un **estado** (*s*), **ACTIONS**(*s*) devuelve un conjunto finito de acciones que se pueden ejecutar.

Decimos que cada una de estas acciones es aplicable en ese estado.

Un ejemplo:



$$\text{ACTIONS}(Arad) = \{ToSibiu, ToTimisoara, ToZerind\}.$$

PROBLEMAS DE BUSQUEDA

2. RESULT:

Un modelo de transición, que describe lo que hace cada acción. **RESULT (*s, a*)** devuelve el estado que resulta de realizar la acción ***a*** en el estado ***s*** . Por ejemplo:

$$\text{RESULT}(Arad, ToZerind) = Zerind.$$

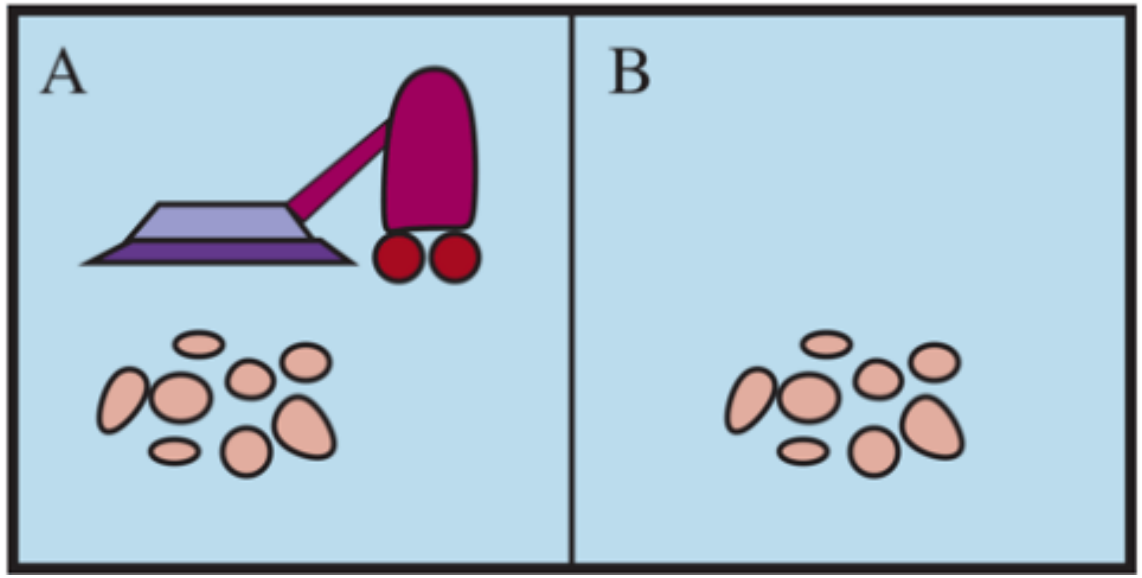
- 3. ACTION-COST: Cuando estamos programando o haciendo matemáticas, buscamos el costo numérico de aplicar la acción ***"a"*** en el estado ***s*** para alcanzar el estado ***s'***. Un agente que resuelve problemas debería usar una función de costo que refleje su propia medida de desempeño.

$$\text{ACTION-COST}(s, a, s')$$

PROBLEMAS DE BUSQUEDA

EXAMPLE: VACOOM PROBLEM

El mundo de una aspiradora, que consiste en un agente de limpieza robótico en un mundo compuesto por cuadrados que pueden estar sucios o limpios. El agente de la aspiradora percibe en qué cuadrado se encuentra y si hay suciedad en el cuadrado. El agente comienza en el cuadrado A.

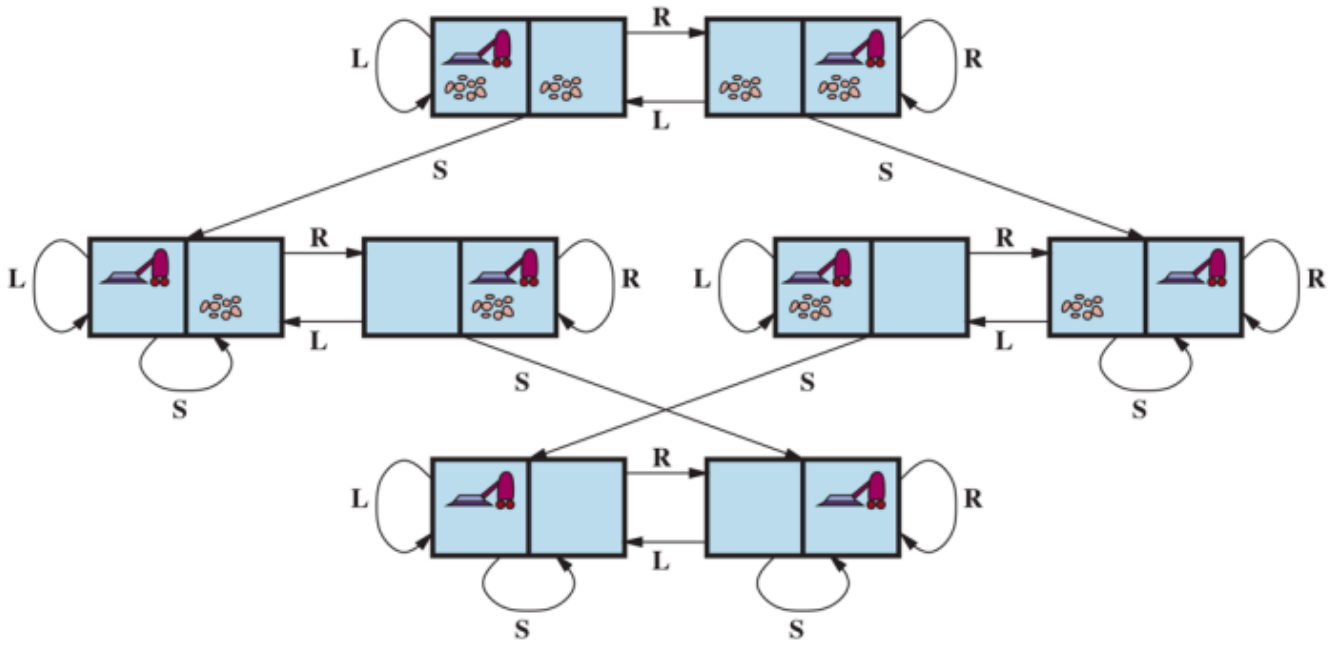


Las **ACTIONS** disponibles son:
moverse a la derecha,
moverse a la izquierda,
aspirar la suciedad o
no hacer nada.

PROBLEMAS DE BUSQUEDA

EXAMPLE: VACOOM PROBLEM

STATES: Un estado del mundo indica qué objetos están en qué celdas. Los objetos son el agente y cualquier suciedad. En la versión simple de dos celdas, el agente puede estar en cualquiera de las dos celdas, y cada celda puede contener suciedad o no , entonces tenemos: 2*2*2=8 **ESTADOS POSIBLES**



The state-space graph for the two-cell vacuum world. There are 8 states and three actions for each state: L = *Left*, R = *Right*, S = *Suck*.

PROBLEMAS DE BUSQUEDA

EXAMPLE: VACOOM PROBLEM

Percept sequence	Action
[A, Clean]	Right
[A, Dirty]	Suck
[B, Clean]	Left
[B, Dirty]	Suck
[A, Clean], [A, Clean]	Right
[A, Clean], [A, Dirty]	Suck
⋮	⋮
[A, Clean], [A, Clean], [A, Clean]	Right
[A, Clean], [A, Clean], [A, Dirty]	Suck
⋮	⋮

function REFLEX-VACUUM-AGENT([*location*,*status*]) **returns** an action

if *status* = *Dirty* **then return** *Suck*
else if *location* = *A* **then return** *Right*
else if *location* = *B* **then return** *Left*

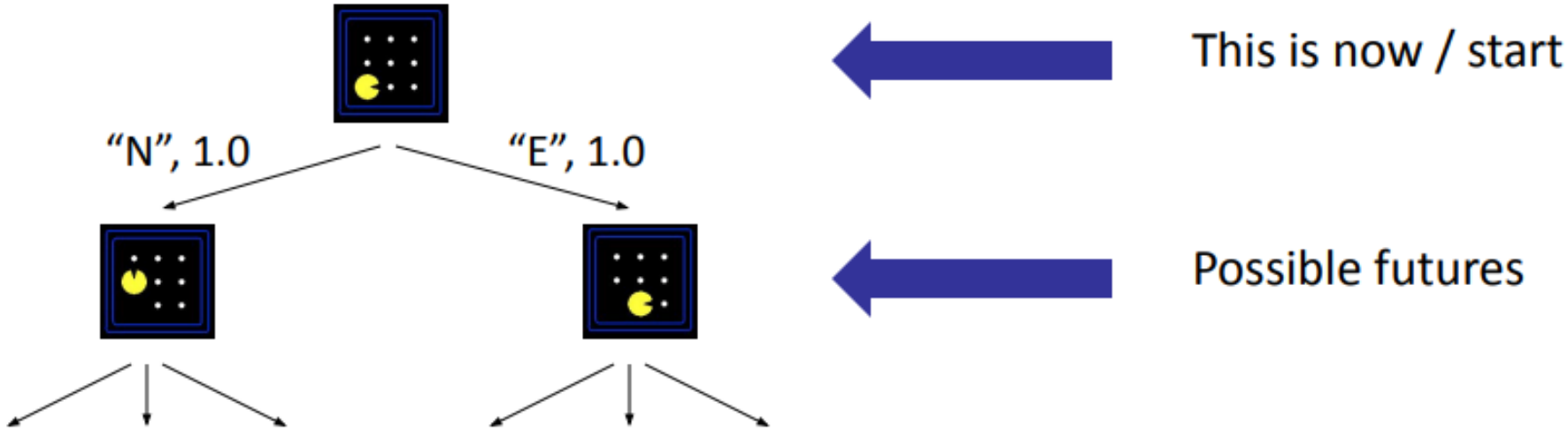
Search Problems

UNIVERSIDAD EAFIT
VIGILADA | MINEDUCACIÓN
Escuela de Ciencias Aplicadas e Ingeniería

CLASS EXERCISE

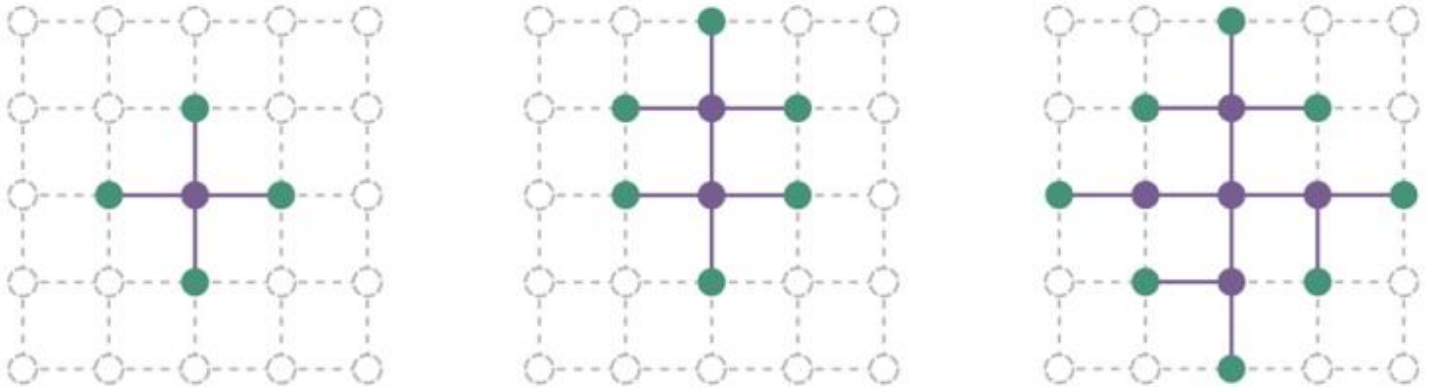
Search trees

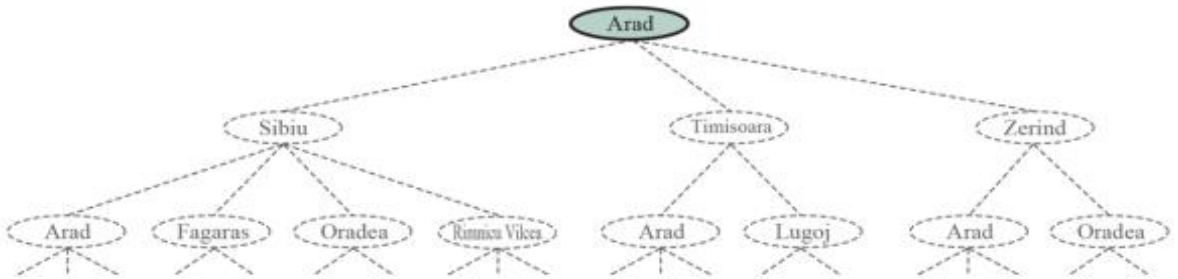
Podemos representar el problema de búsqueda como una estructura de árbol, donde cada nodo en el árbol de búsqueda corresponde a un estado en el espacio de estados y los arcos en el árbol de búsqueda corresponden a acciones. La raíz del árbol corresponde al estado inicial del problema.

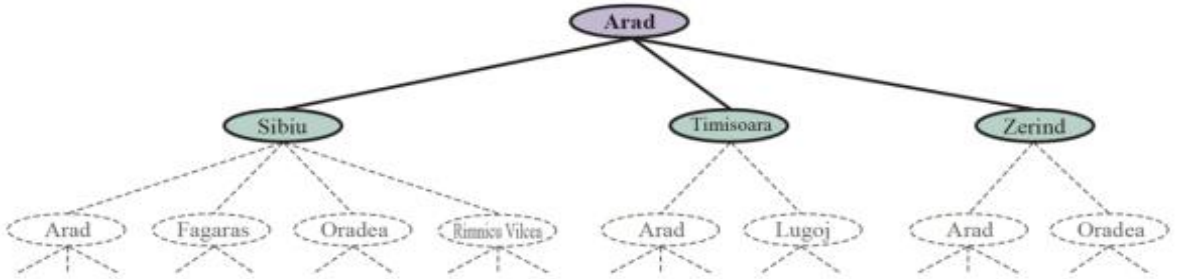


Search trees

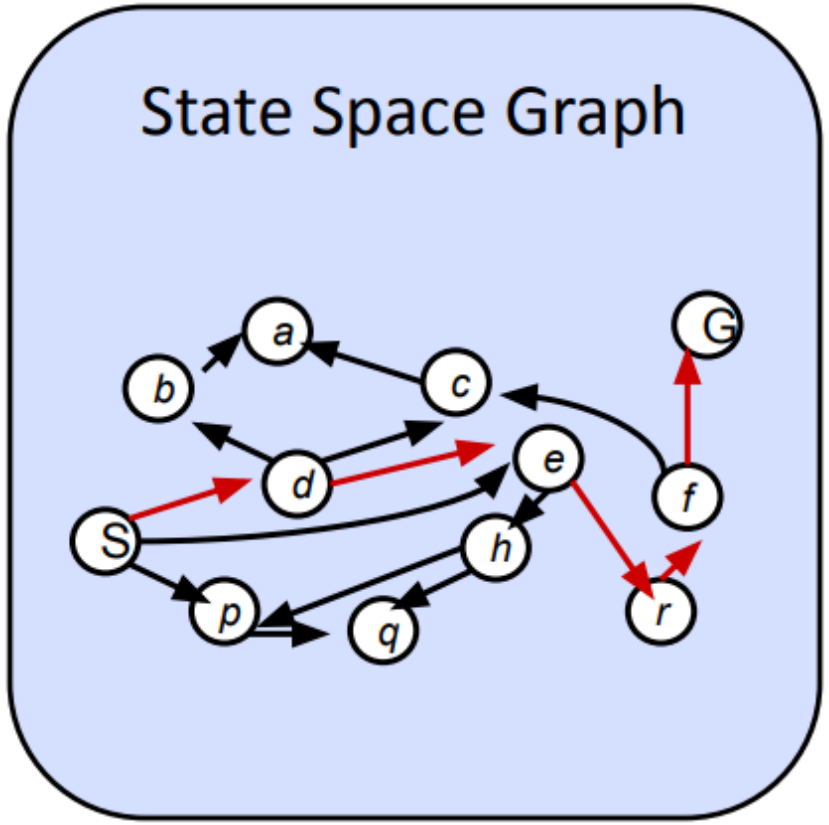
La frontera es el conjunto de nodos (y los estados correspondientes) que han sido alcanzados pero aún no expandidos; el interior es el conjunto de nodos que han sido expandidos; y el exterior es el conjunto de estados que no han sido alcanzados.





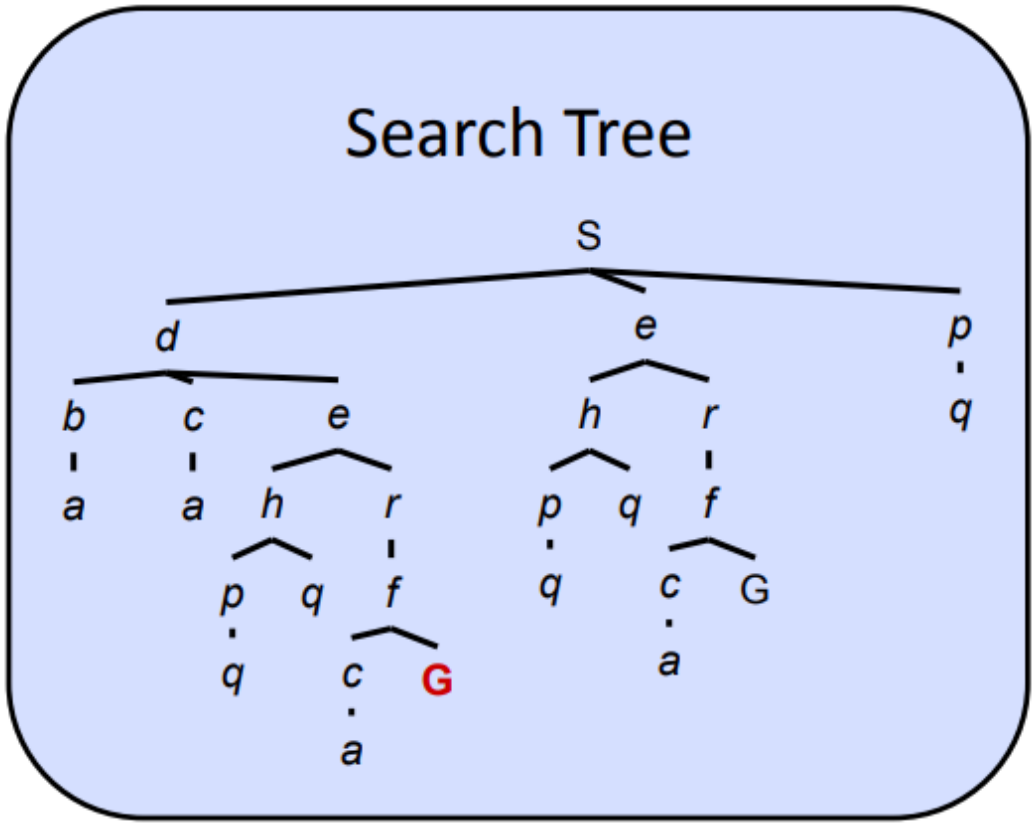


Search trees



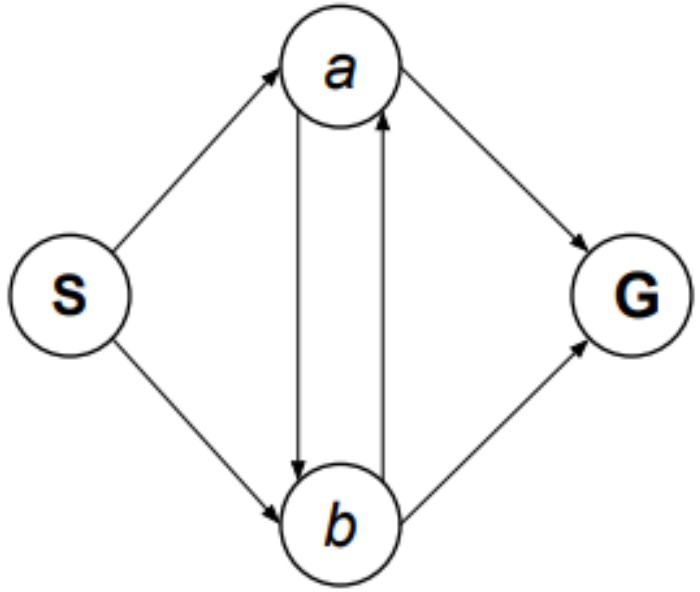
Each NODE in in the search tree is an entire PATH in the state space graph.

We construct both on demand – and we construct as little as possible.



Search trees

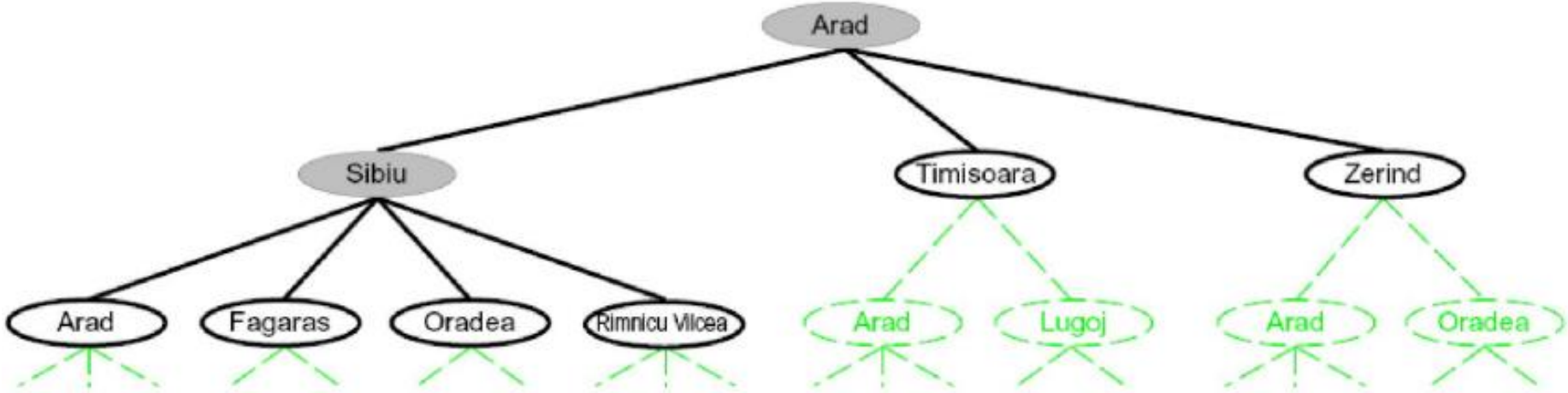
Consider this 4-state graph:



How big is its search tree (from S)?

Search trees

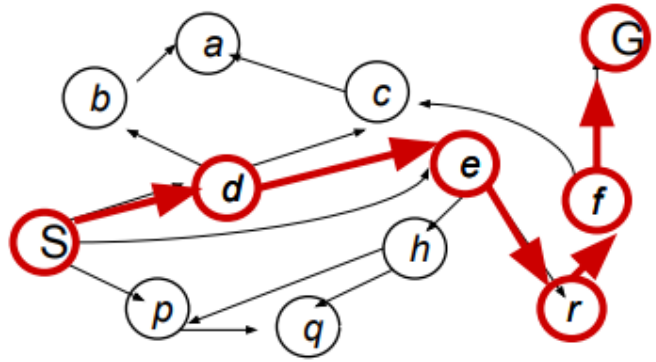
Como funciona la búsqueda en árbol?



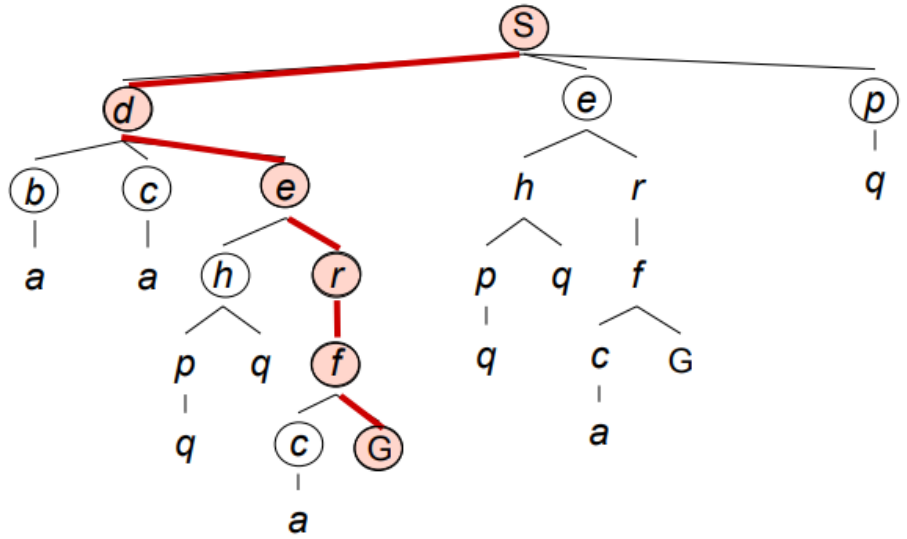
- Search:
  - Expand out potential plans (tree nodes)
  - Maintain a fringe of partial plans under consideration
  - Try to expand as few tree nodes as possible

Search trees


```
function TREE-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
```



- Important ideas:
  - Fringe
  - Expansion
  - Exploration strategy

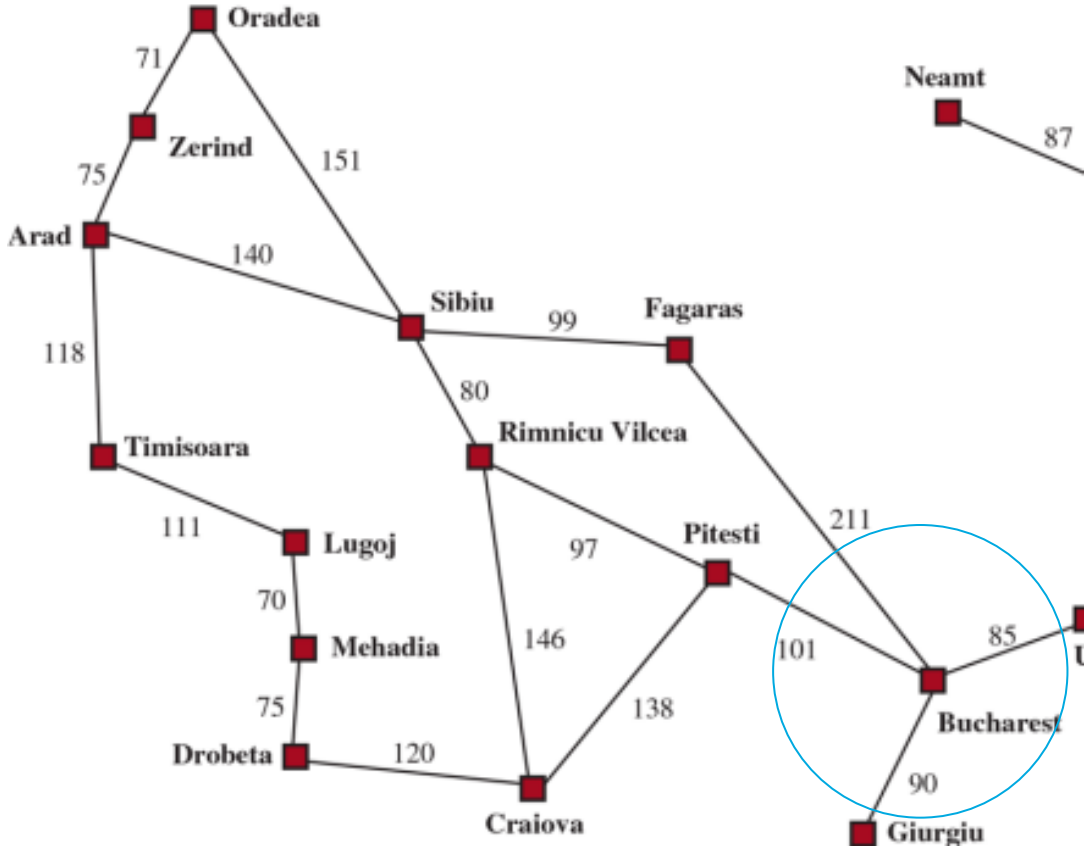


~~s~~
~~s □ d~~
s □ e
s □ p
s □ d □ b
s □ d □ c
~~s □ d □ e~~
s □ d □ e □ h
~~s □ d □ e □ r~~
~~s □ d □ e □ r □ f~~
s □ d □ e □ r □ f □ c
~~s □ d □ e □ r □ f □ G~~

- Main question: which fringe nodes to explore?

Estrategias de búsqueda Ciega

Un algoritmo de búsqueda no informada (ciego), no recibe ninguna pista sobre qué tan cerca está un estado del estado(s) objetivo.



Estrategias de búsqueda Ciega

PUNTOS PRELIMINARES PARA TENER EN CUENTA:

Dado un estado (*s*), **ACTIONS**(*s*) devuelve un conjunto finito de acciones que se pueden ejecutar.

$$\text{ACTIONS}(Arad) = \{ToSibiu, ToTimisoara, ToZerind\}.$$

Tenemos un modelo de transición, que describe lo que hace cada acción. **RESULT**(*s*, *a*) devuelve el estado que resulta de realizar la acción a en el estado s .

$$\text{RESULT}(Arad, ToZerind) = Zerind.$$

Y una función de costo asociada a la acción

$$\text{ACTION-COST}(s, a, s')$$

Estrategias de búsqueda Ciega

Los algoritmos de búsqueda requieren una estructura de datos para mantener un registro del árbol de búsqueda.

Como definimos cada Nodo del árbol desde nuestro código?

- **node.STATE**: el estado al que corresponde el nodo.
- **node.PARENT**: el nodo en el árbol que generó este nodo.
- **node.ACTION**: la acción que se aplicó al estado del padre para generar este nodo.
- **node.PATH-COST**: el costo total del camino desde el estado inicial hasta este nodo.

Estrategias de búsqueda Ciega

Necesitamos una estructura de datos para almacenar la frontera.

La elección adecuada es una cola (**queue**) de algún tipo, porque las operaciones en una frontera son:

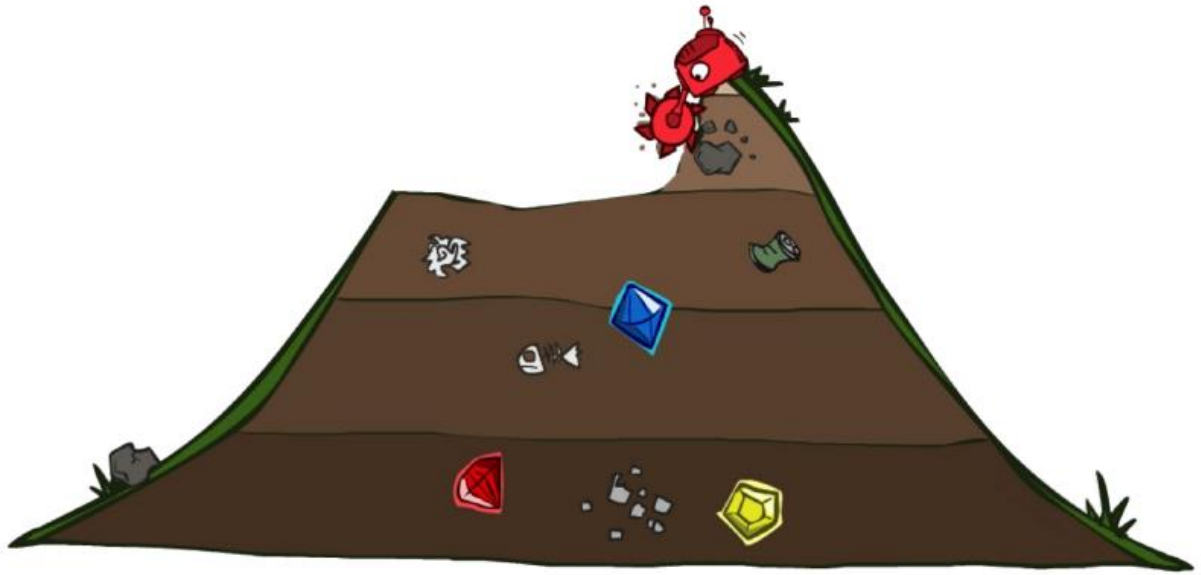
- **IS-EMPTY(frontier)**: devuelve verdadero solo si no hay nodos en la frontera.
- **POP(frontier)**: elimina el nodo superior de la frontera y lo devuelve.
- **TOP(frontier)**: devuelve (pero no elimina) el nodo superior de la frontera.
- **ADD(node, frontier)**: inserta el nodo en su lugar adecuado en la cola.

Estrategias de búsqueda Ciega
Best-First Search

UNIVERSIDAD EAFIT | Escuela de Ciencias Aplicadas e Ingeniería

Cuando las acciones tienen costos diferentes, una elección obvia es usar la búsqueda del mejor primero (**best-first search**) donde la función de evaluación es el costo del camino desde la raíz hasta el nodo actual. Esto es conocido como **búsqueda de costo uniforme** en la comunidad de IA.

Uniform Cost Search



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

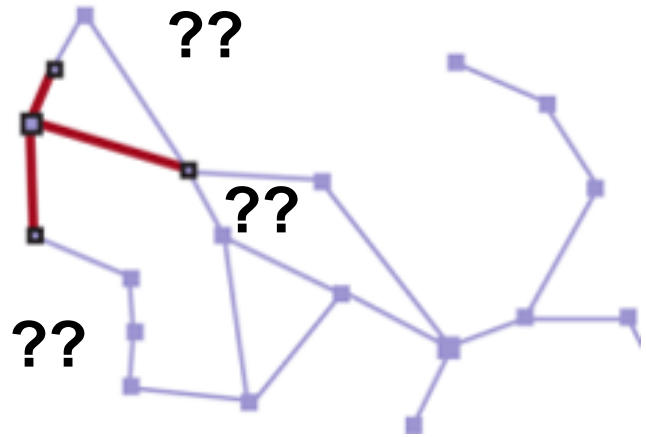
Estrategias de búsqueda Ciega
Best First Search

UNIVERSIDAD EAFIT
Escuela de Ciencias Aplicadas e Ingeniería

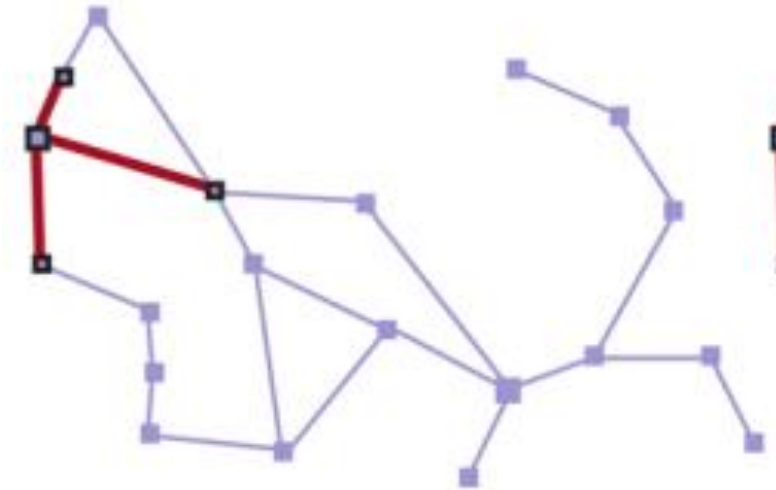
```
function BEST-FIRST-SEARCH(problem, f) returns a solution node or failure
  node ← NODE(STATE=problem.INITIAL)
  frontier ← a priority queue ordered by f, with node as an element
  reached ← a lookup table, with one entry with key problem.INITIAL and value node
  while not IS-EMPTY(frontier) do
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

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

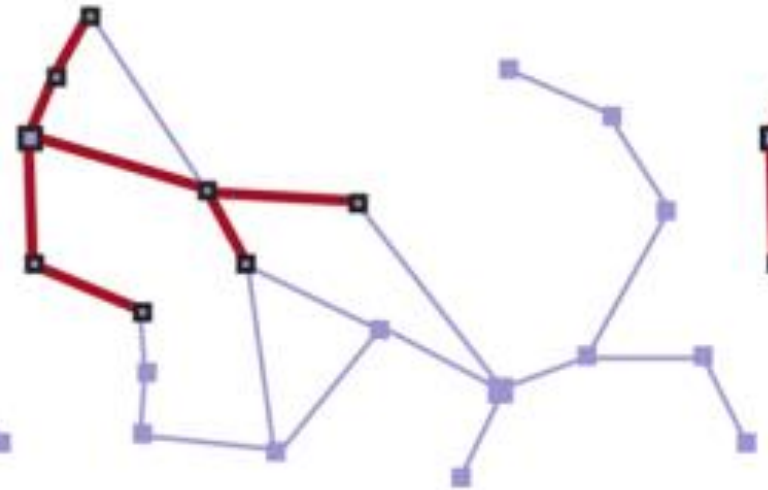
El algoritmo devuelve una indicación de fallo o un nodo que representa un camino hacia el **Goal State**

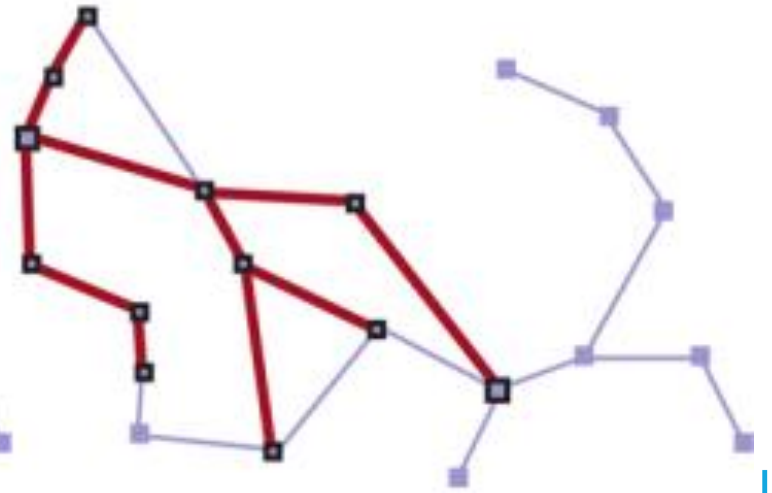
Estrategias de búsqueda Ciega
Best First Search



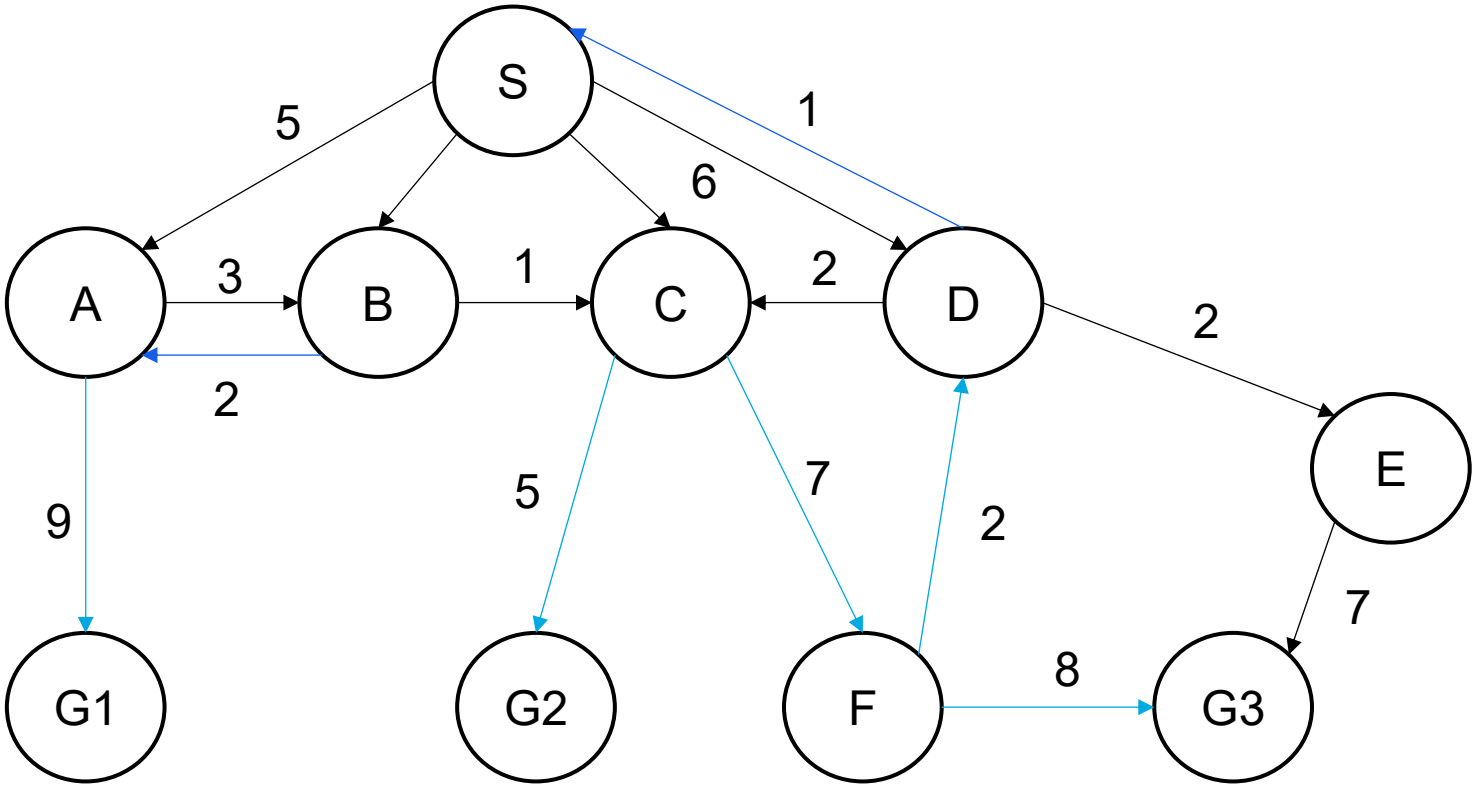
En cada iteración, elegimos un nodo en la frontera con el valor mínimo de $f(n)$, lo devolvemos si su estado es un estado objetivo, y de lo contrario aplicamos una función de expansión.



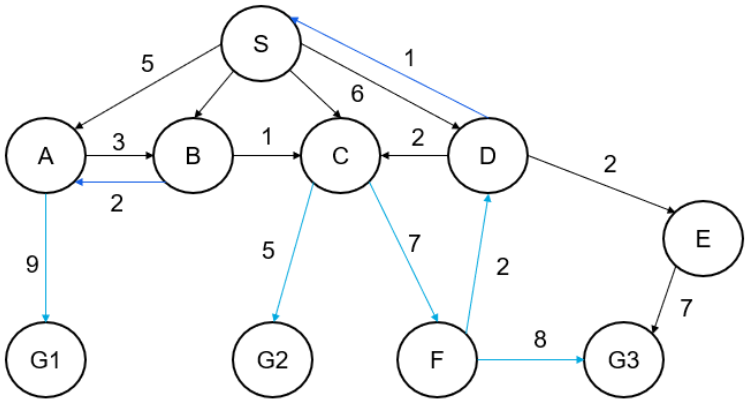




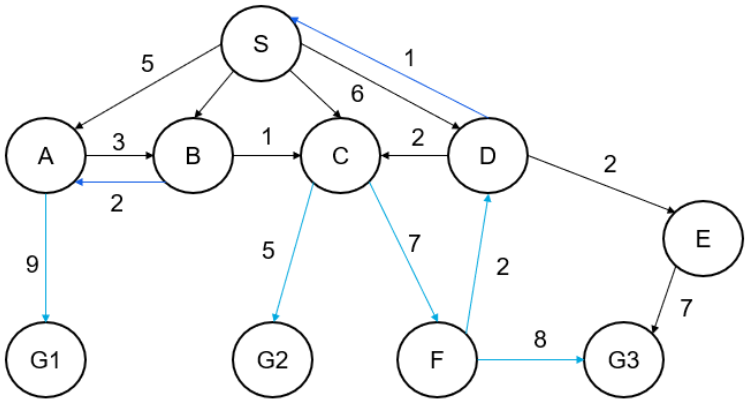
Ejemplo



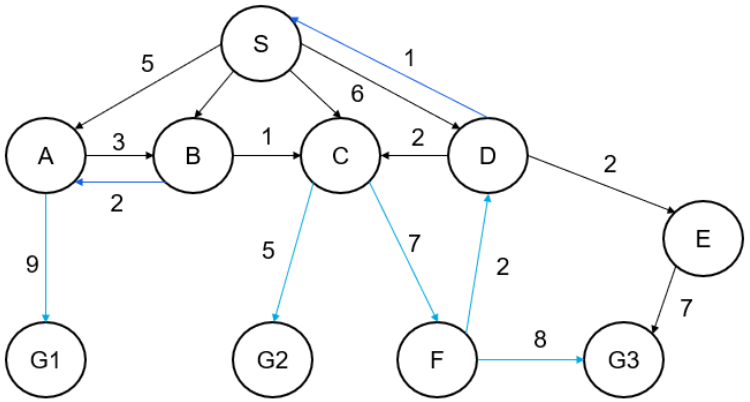
Ejemplo



Ejemplo



Ejemplo



Estrategias de búsqueda Ciega

CODE AND CLASS EXERCISE

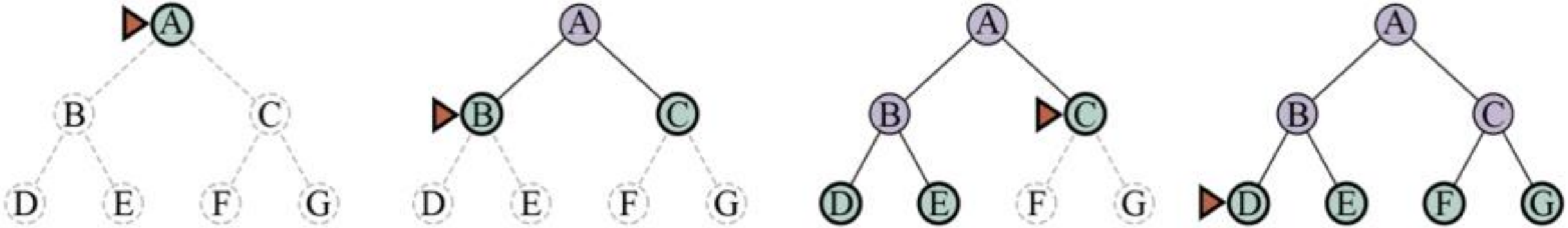
Estrategias de búsqueda Ciega
Breadth-First Search

UNIVERSIDAD EAFIT
Escuela de Ciencias Aplicadas e Ingeniería

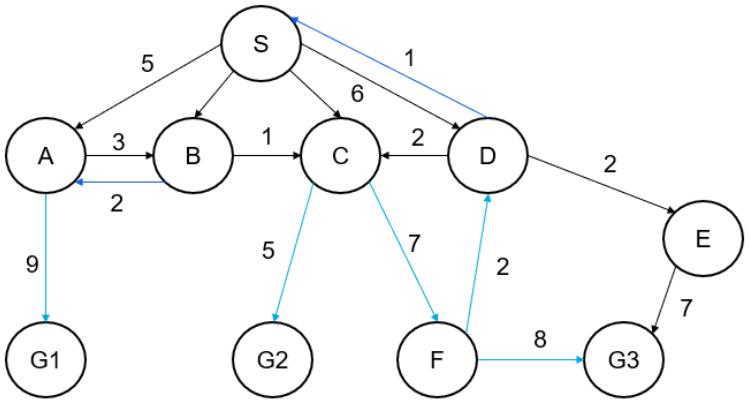
Cuando todas las acciones tienen el mismo costo, una estrategia adecuada es la **Breadth-first Search**, en la que primero se expande el nodo raíz, luego se expanden todos los sucesores del nodo raíz, luego los sucesores de estos, y así sucesivamente.

Estrategias de búsqueda Ciega-Breadth-First Search

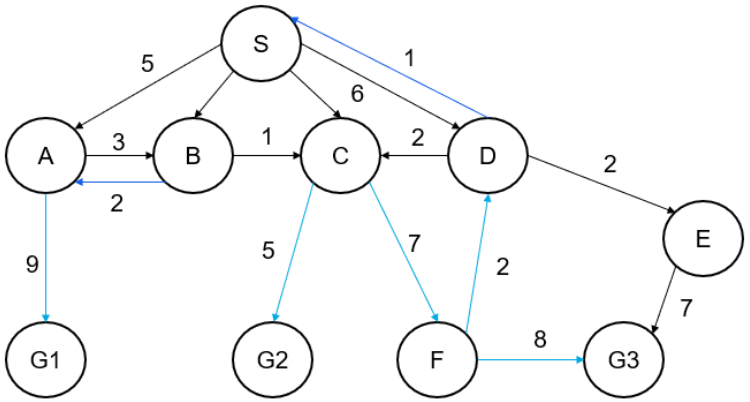
La búsqueda siempre encuentra una solución con un número mínimo de acciones, porque cuando está generando nodos en la profundidad d, ya ha generado todos los nodos en la profundidad d-1 por lo que, si uno de ellos fuera una solución, habría sido encontrada.



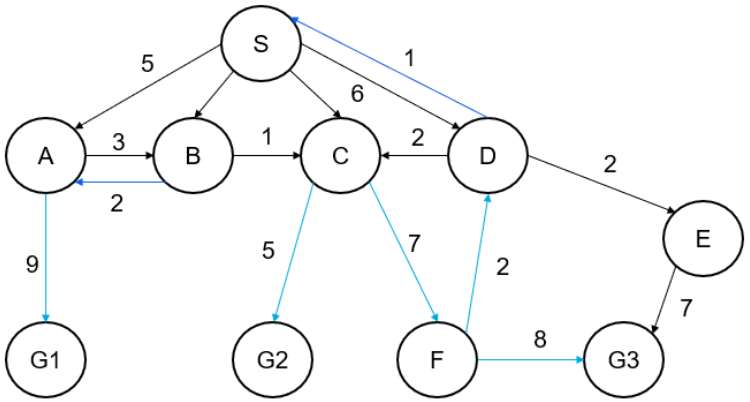
Ejemplo



Ejemplo



Ejemplo



Estrategias de búsqueda Ciega- Breadth-First Search

UNIVERSIDAD EAFIT | Escuela de Ciencias Aplicadas e Ingeniería

```
function BREADTH-FIRST-SEARCH(problem) returns a solution node or failure
  node ← NODE(problem.INITIAL)
  if problem.IS-GOAL(node.STATE) then return node
  frontier ← a FIFO queue, with node as an element
  reached ← {problem.INITIAL}
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    for each child in EXPAND(problem, node) do
      s ← 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
```

'reached' puede ser un conjunto de estados en lugar de un mapeo de estados a nodos, porque una vez que hemos alcanzado un estado, nunca podremos encontrar un mejor camino hacia ese estado.

Esto también significa que podemos hacer una prueba temprana del objetivo.

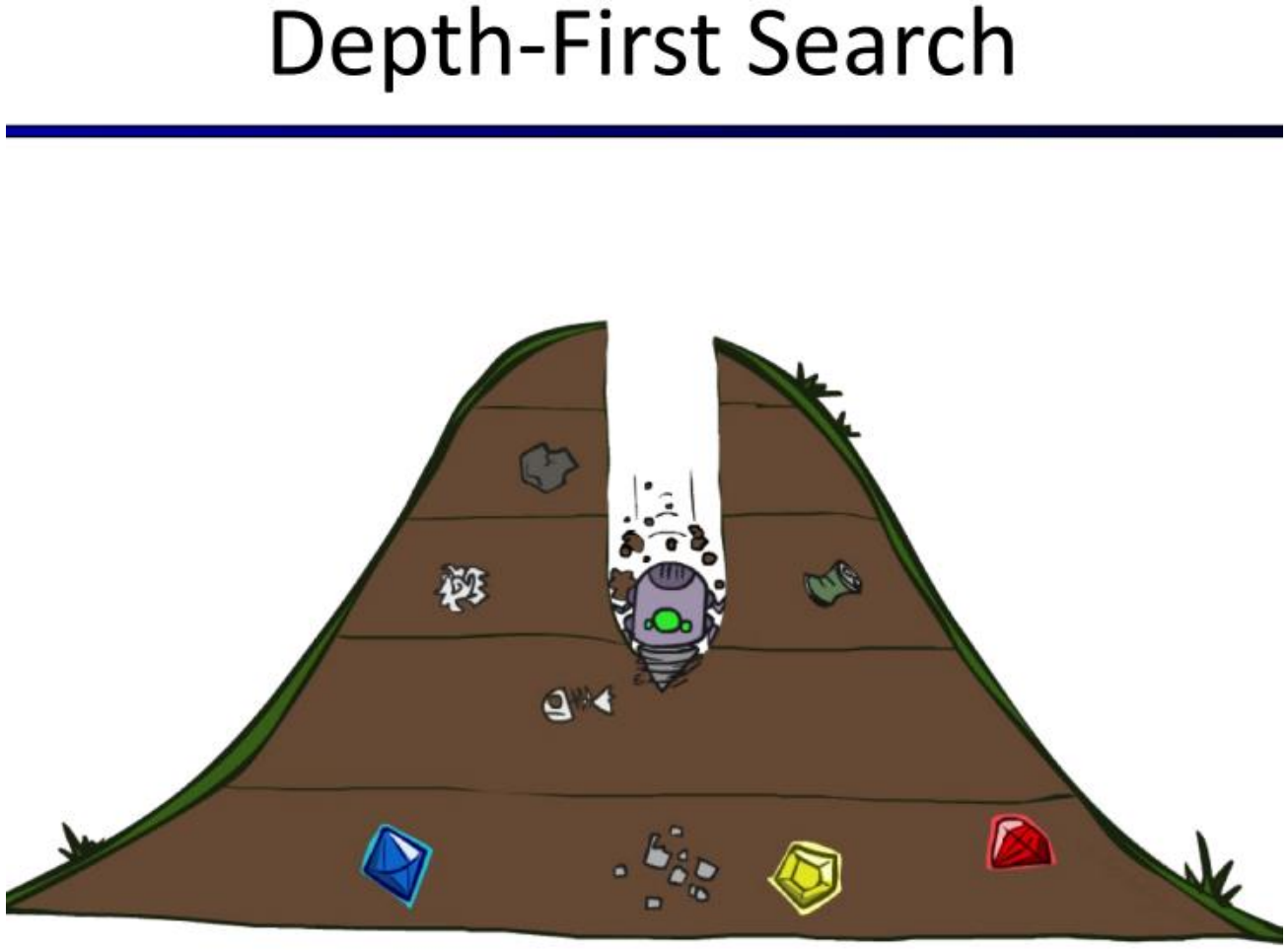
Estrategias de búsqueda Ciega

CODE AND CLASS EXERCISE

Estrategias de búsqueda Ciega
Depth-First Search

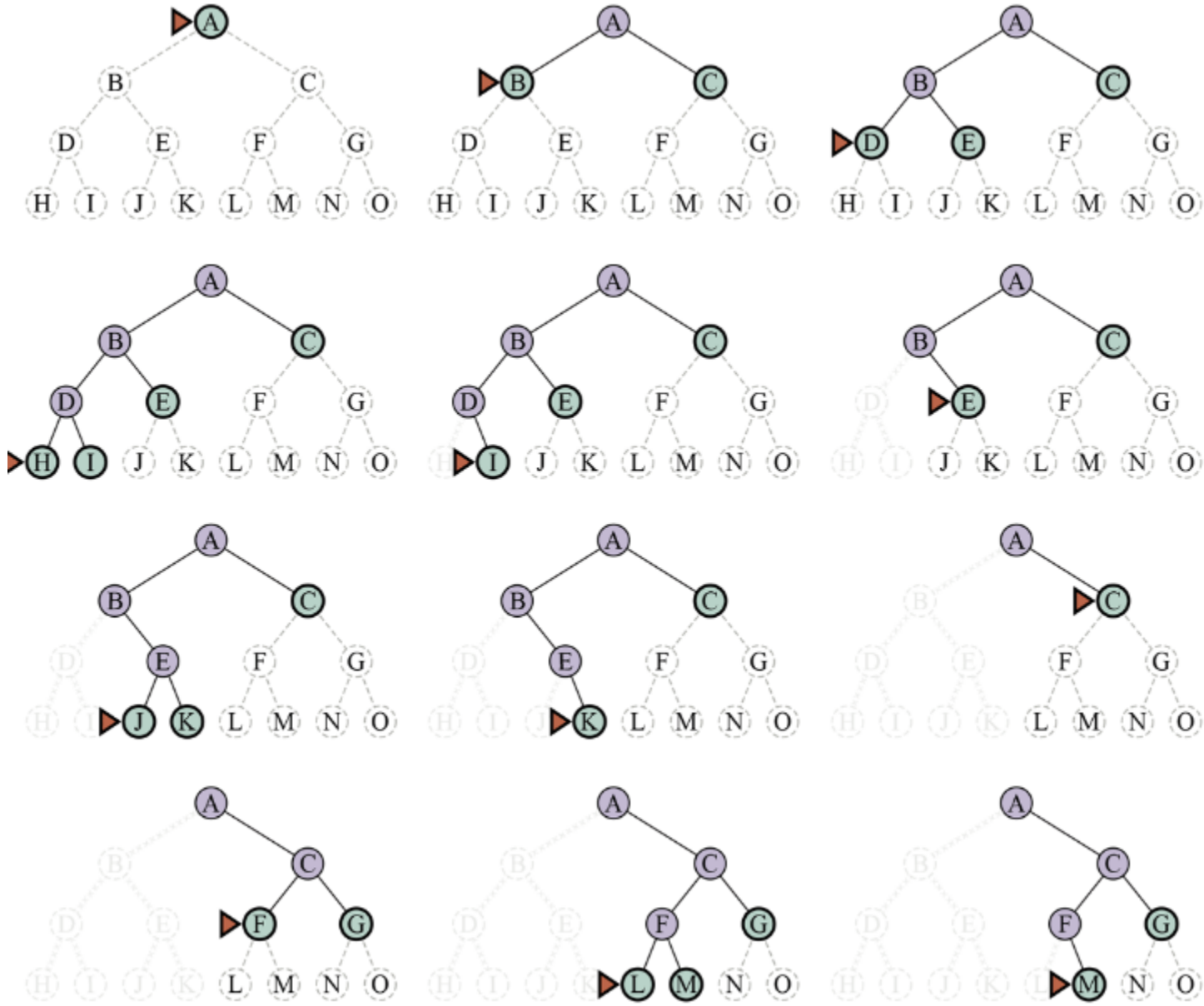
UNIVERSIDAD EAFIT
Escuela de Ciencias Aplicadas e Ingeniería

La búsqueda en profundidad **(Depth-First Search)** siempre expande primero el nodo más profundo de la frontera.

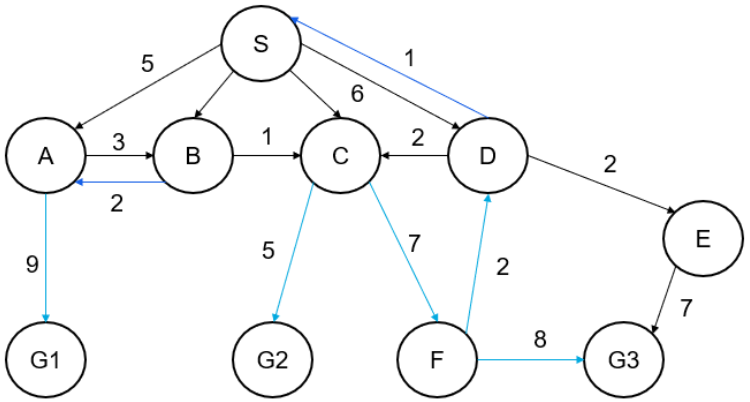


Estrategias de búsqueda Ciega
Depth-First Search

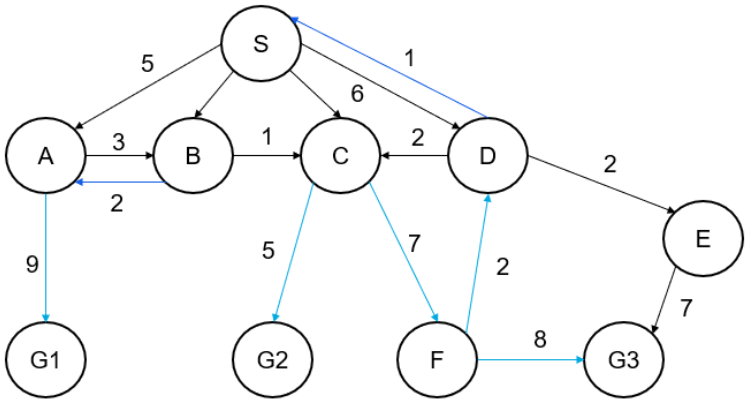
La búsqueda avanza al nivel más profundo del árbol de búsqueda, donde los nodos no tienen sucesores. Luego, la búsqueda «retrocede» al siguiente nodo más profundo que aún tiene sucesores sin expandir.



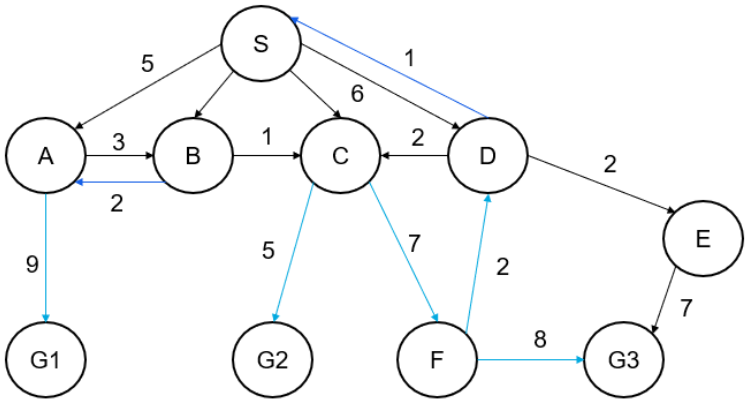
Ejemplo



Ejemplo



Ejemplo



Estrategias de búsqueda Ciega
Depth-limited-Search


```
function ITERATIVE-DEEPENING-SEARCH(problem) returns a solution node or failure
  for depth = 0 to ∞ do
    result ← DEPTH-LIMITED-SEARCH(problem, depth)
    if result ≠ cutoff then return result

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

Tareas

- Teniendo en cuenta las bases vistas en clase, investigar en que consiste la técnica de Iterative Deepening search
- Estudiar el Notebook de Iterative Deepening search suministrado por el profesor
- Investigar en que consisten las métricas de rendimiento de los algoritmos de búsqueda y como pueden ser estimadas: completeness,cost optimality,time complexity,space complexity y como se aplican a los algoritmos vistos en clase

Referencias

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

CS 188 Fall 2025, Emma Pierson, Peyrin Kao
https://inst.eecs.berkeley.edu/~cs188/fa25/