UNIVERSIDAD EAFIT
VIGILADA | MINEDUCACIÓN

Escuela de
Ciencias Aplicadas
e Ingeniería

Artificial Intelligence

Lecture02 – Problem Formulation and Uninformed Search

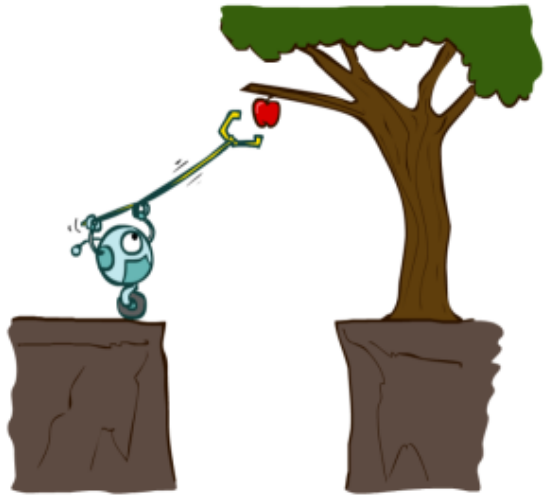
Contenido

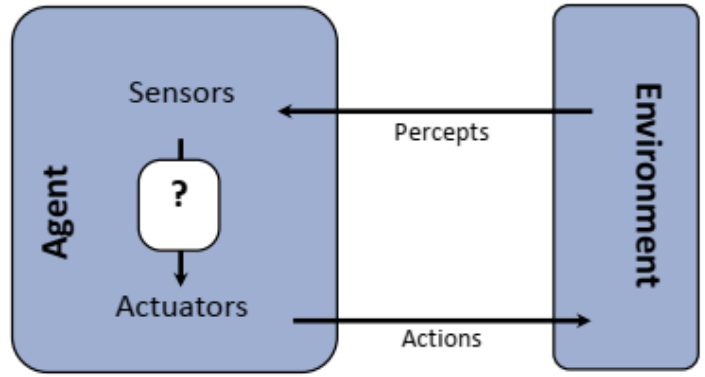
UNIVERSIDAD EAFIT
Escuela de Ciencias Aplicadas e Ingeniería

1. Agentes de resolución de problemas
2. Formulación de problemas
3. Solución de problemas
4. Problemas de búsqueda
5. Estrategias de Búsqueda Ciega

Agentes de Resolución de Problemas

Cuando la acción correcta a tomar no es inmediatamente obvia, un agente puede necesitar planificar con anticipación: considerar una secuencia de acciones que formen un camino hacia un estado objetivo. A tal agente se le llama **Agente de resolución de problemas**, y el proceso computacional que lleva a cabo se llama **búsqueda**.

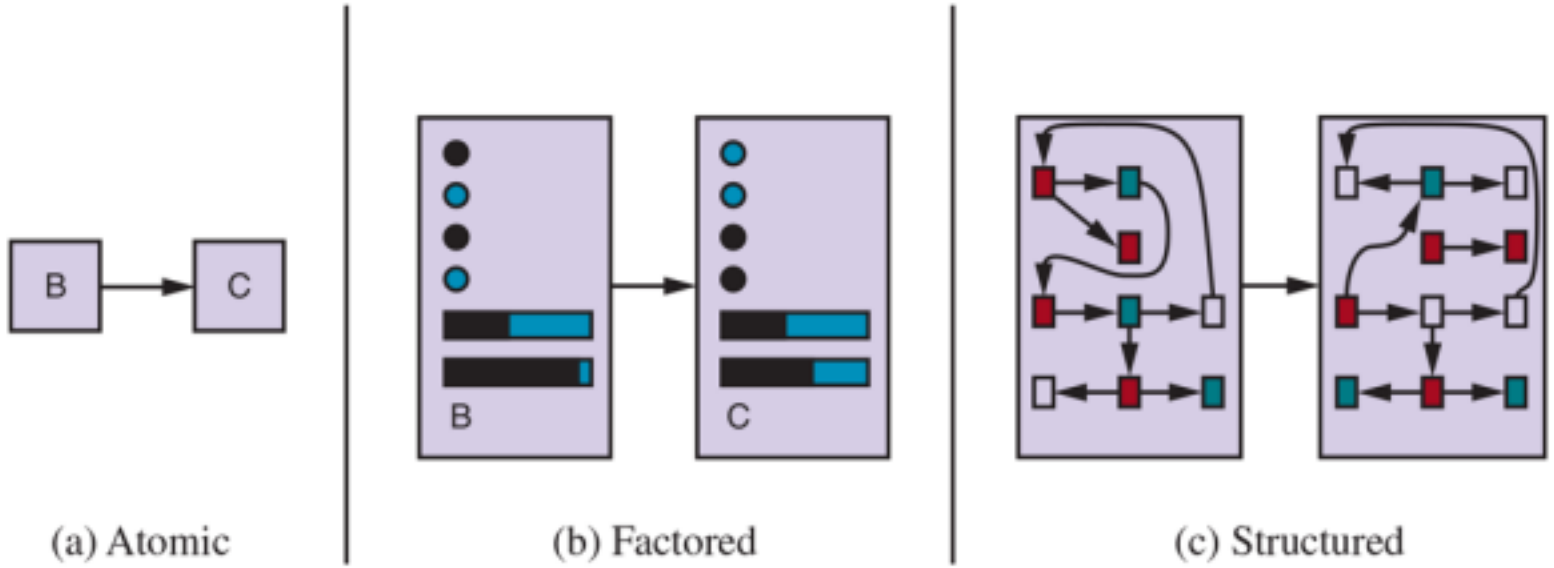




Agentes de Resolución de Problemas

UNIVERSIDAD EAFIT
Escuela de Ciencias Aplicadas e Ingeniería

- Los agentes de resolución de problemas utilizan representaciones atómicas.

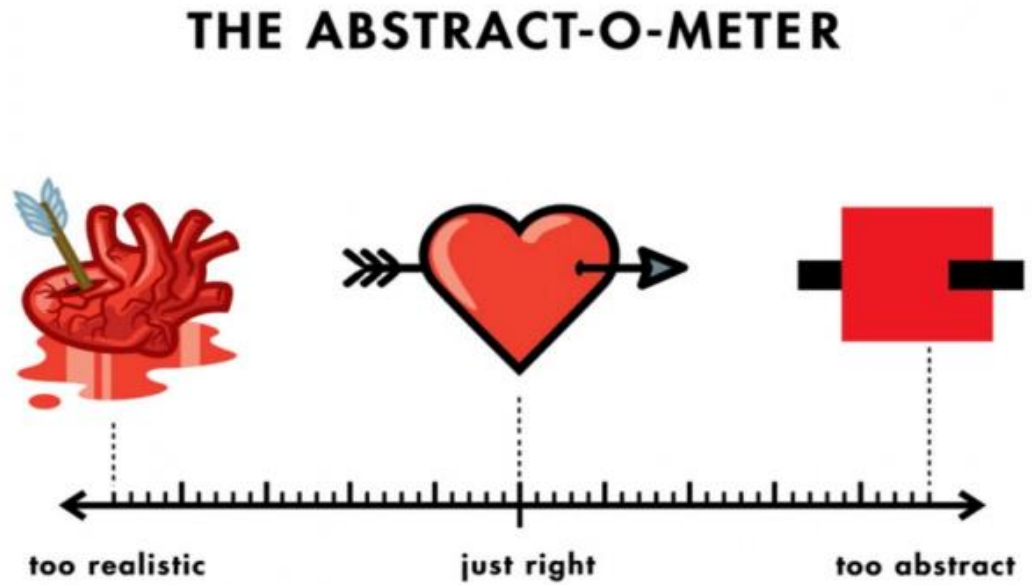


Agentes de Resolución de Problemas

UNIVERSIDAD EAFIT | Escuela de Ciencias Aplicadas e Ingeniería

Para iniciar con el estudio de algoritmos de búsqueda consideraremos los entornos simples: episódicos, de un solo agente, completamente observables, deterministas, estáticos, discretos y conocidos.

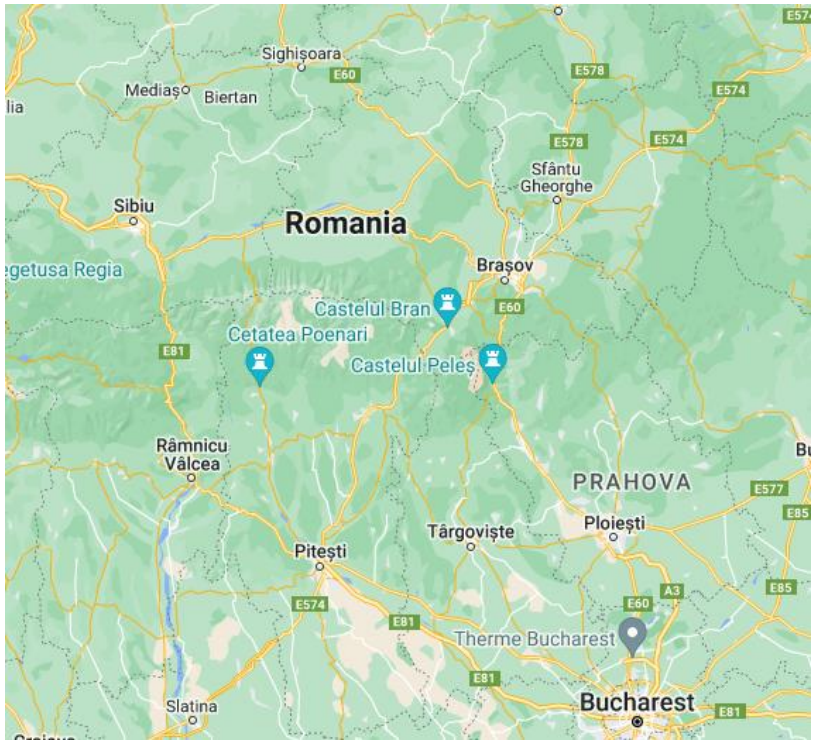
Trabajaremos un poco de… Abstracción…



Formulación de problemas

Imagina a un agente ubicado en Rumania. supongamos que el agente está actualmente en la ciudad de Arad y tiene un boleto no reembolsable para volar desde Bucarest al día siguiente.

El agente observa las señales de tráfico y ve que hay tres caminos que salen de Arad: uno hacia Sibiu, otro hacia Timisoara y otro hacia Zerind. Ninguno de estos es el destino, por lo que, a menos que el agente esté familiarizado con la geografía de Rumania, no sabrá qué camino seguir.



Formulación de problemas

- Nuestra formulación del problema de llegar a Bucarest es un modelo, una descripción matemática abstracta, y no la realidad.
- Si el agente no tiene información adicional, es decir, si el entorno es desconocido, entonces el agente no puede hacer nada mejor que ejecutar una de las acciones al azar.
- Sin embargo, si proporcionamos información del mundo el agente puede seguir este proceso de resolución de problemas en cuatro fases:

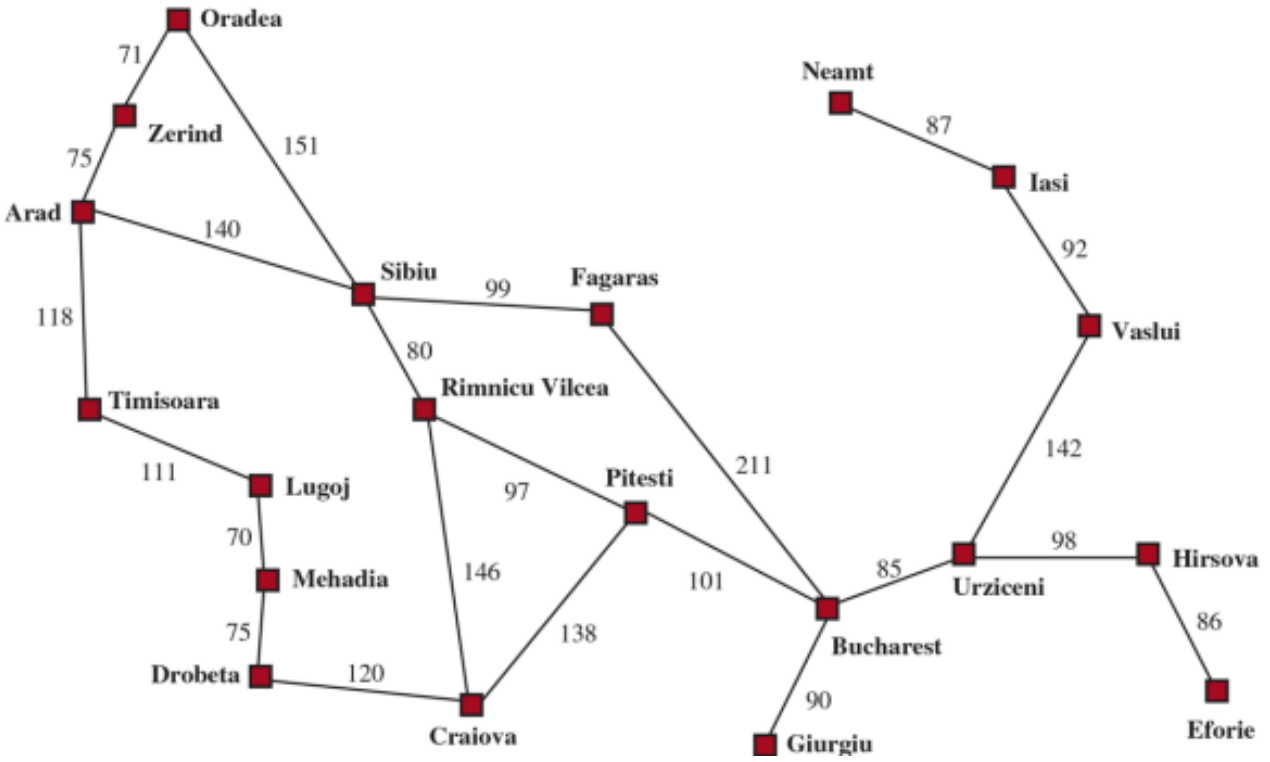
Formulación de problemas

- **PASO 1: Definición del objetivo**

El agente adopta el objetivo de llegar a Bucarest. Los objetivos organizan el comportamiento al limitar los fines y, por ende, las acciones a considerar.

- **PASO 2: Formulación**

El agente elabora una descripción de los estados y acciones necesarios para alcanzar el objetivo, es decir, un modelo abstracto de la parte relevante del mundo.



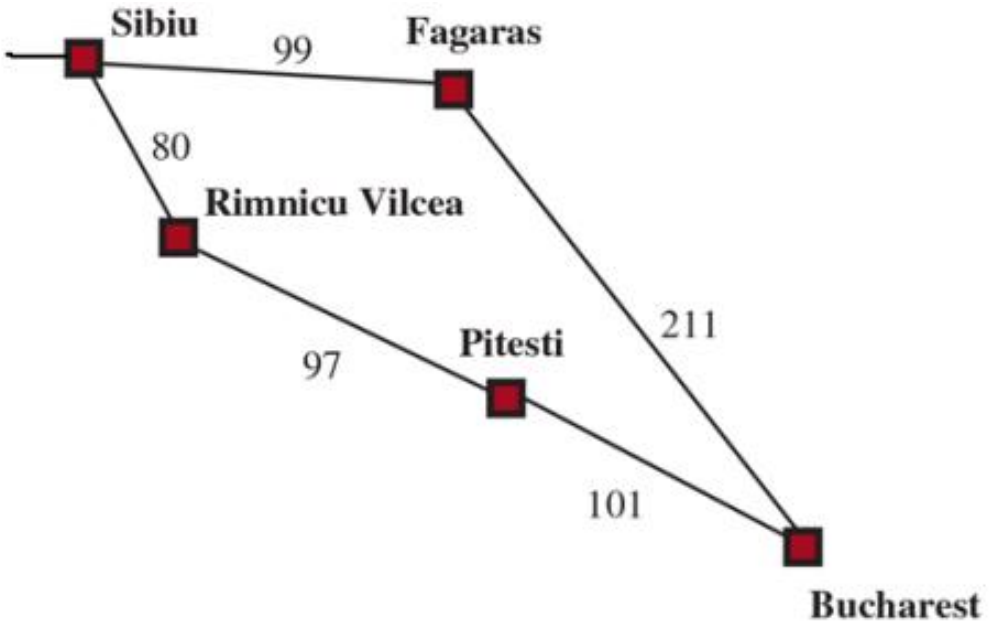
Formulación de problemas

- **PASO 3 BÚSQUEDA:**

Antes de tomar cualquier acción en el mundo real, 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 ignorar sus percepciones mientras ejecuta las acciones (COMO SIGUIENDO LA SOLUCIÓN A OJOS CERRADOS) porque la solución está garantizada para llevar al objetivo.



PROBLEMAS DE BUSQUEDA

Entonces… Como podemos definir un problema?

Un conjunto de posibles **estados** en los que el entorno puede estar. A esto lo llamamos el **espacio de estados.**

El estado inicial en el que el agente comienza. Por ejemplo: Arad.

Un conjunto de uno o más estados objetivo. A veces hay un único **estado objetivo** (por ejemplo, Bucarest), a veces hay un pequeño conjunto de estados objetivo alternativos

Las **acciones** disponibles para el agente.

PROBLEMAS DE BUSQUEDA

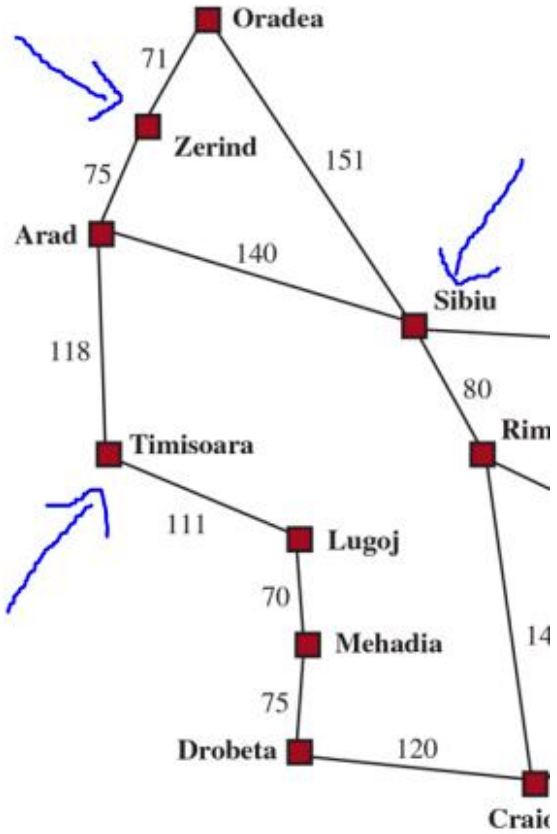
COMPOSICIÓN:

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:



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

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:

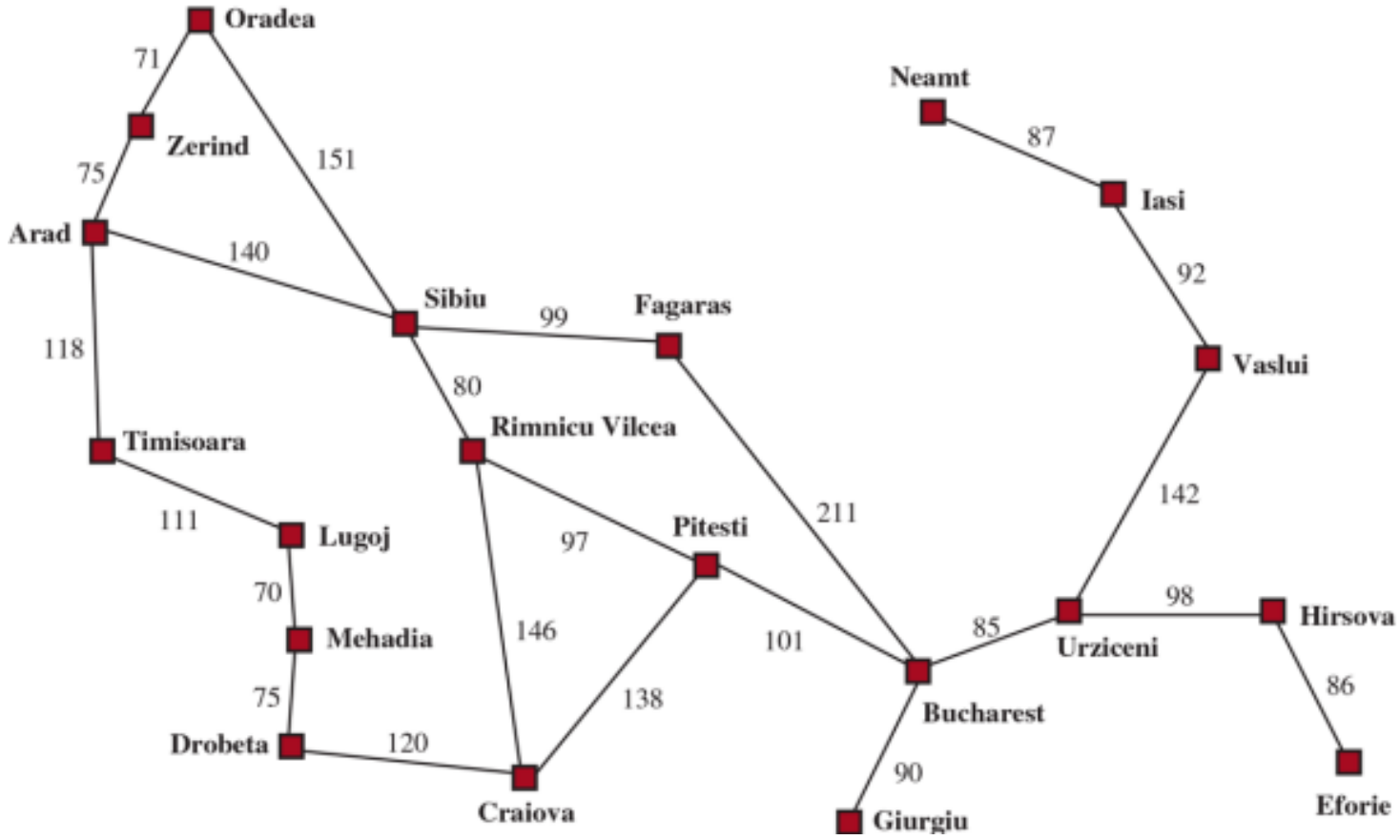
$$\text{RESULT}(Arad, ToZerind) = Zerind.$$

- 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

El espacio de estados puede representarse como un grafo en el que los vértices son estados y los arcos dirigidos entre ellos son acciones. El mapa de Rumania mostrado en la figura es un grafo de este tipo, donde cada carretera indica dos acciones, una en cada dirección.

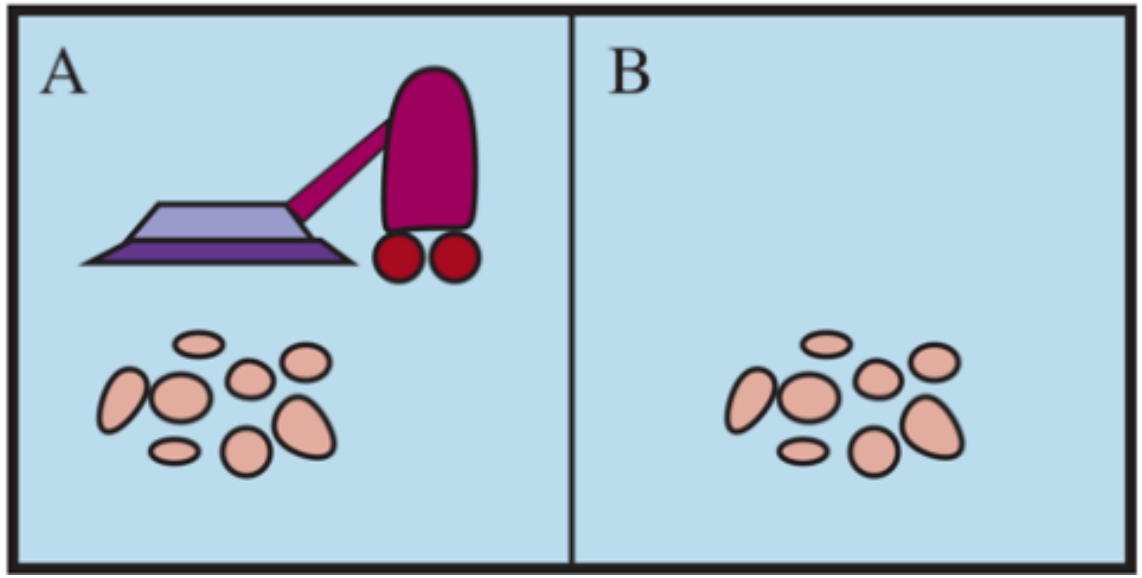


PROBLEMAS DE BUSQUEDA

UNIVERSIDAD EAFIT | Escuela de Ciencias Aplicadas e Ingeniería

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.

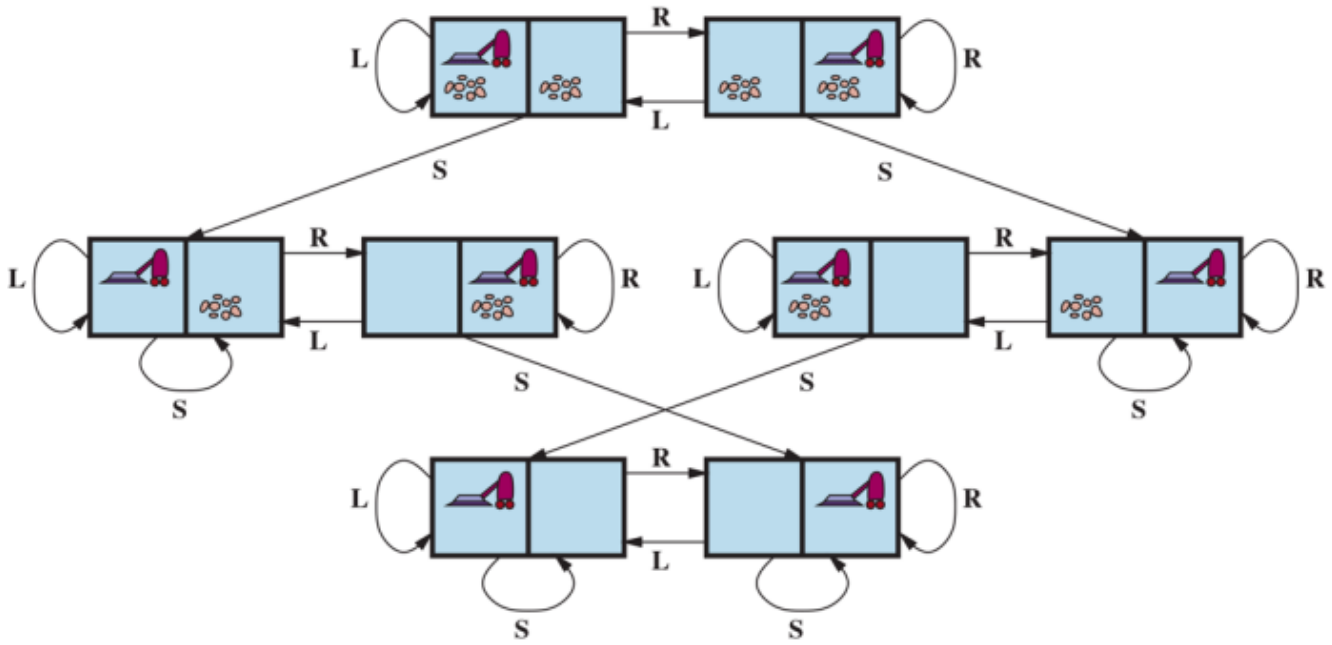


Las **ACTIONS** disponibles son:
moverse a la derecha,
moverse a la izquierda,
aspirar la suciedad o
no hacer nada.

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*

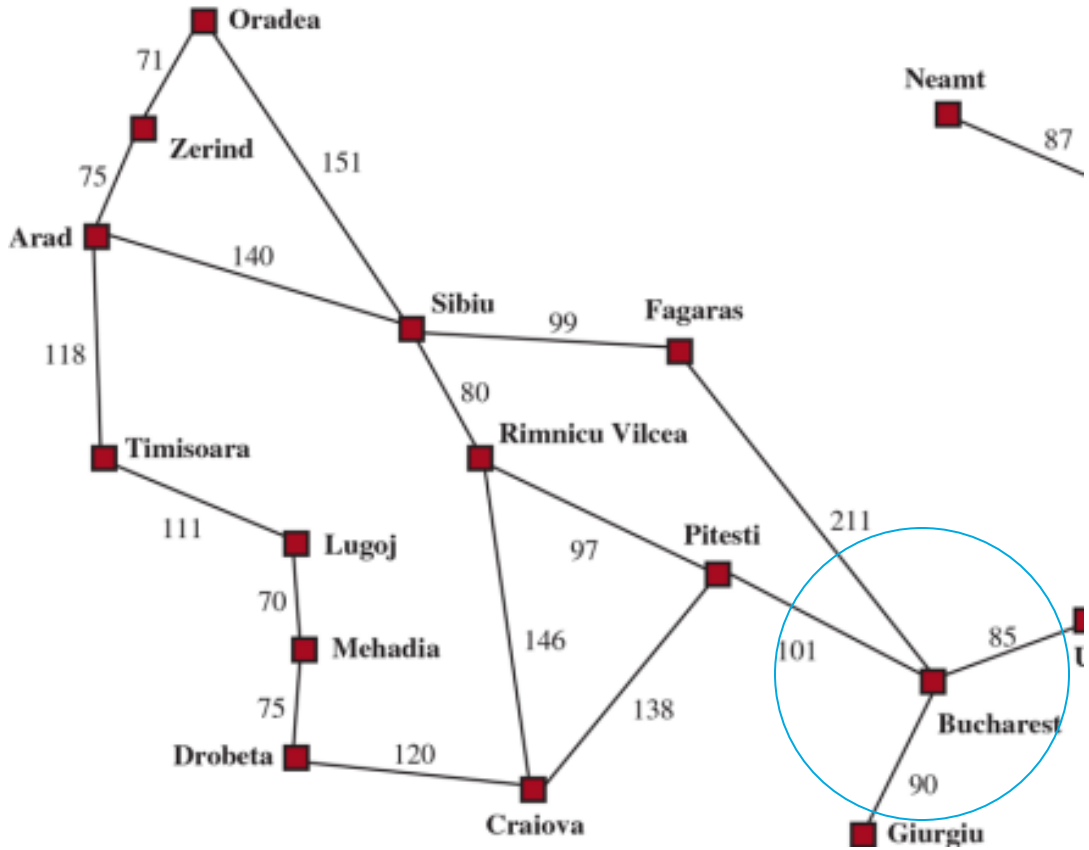
Search Problems

UNIVERSIDAD EAFIT
Escuela de Ciencias Aplicadas e Ingeniería

CLASS EXERCISE

Estrategias de búsqueda Ciega

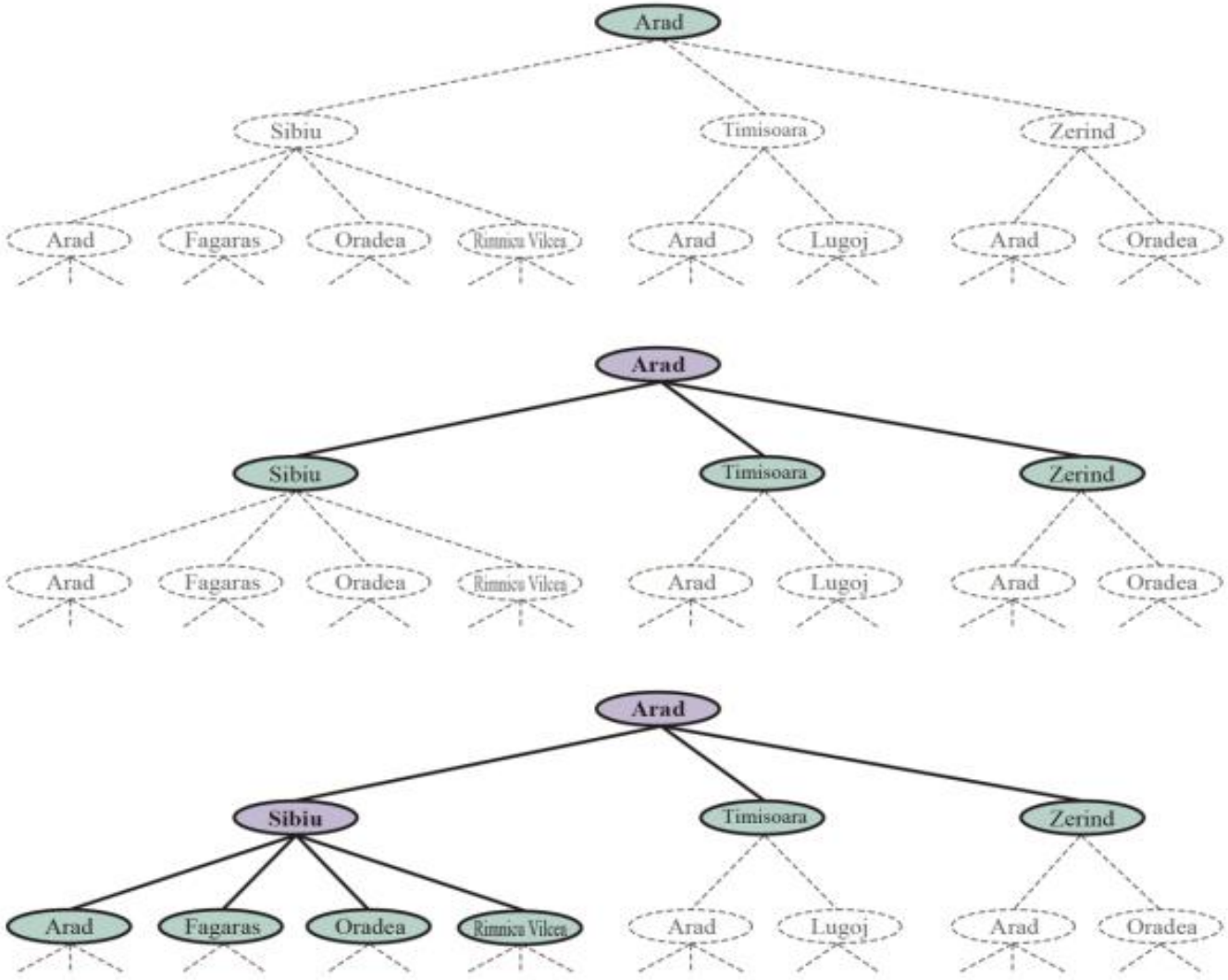
Un algoritmo de búsqueda no informada (ciego), no recibe ninguna pista sobre qué tan cerca está un estado del objetivo o los objetivos. Por ejemplo, considera nuestro agente en Arad con el objetivo de llegar a Bucarest. Un agente no informado, sin conocimiento de la geografía rumana, no tiene idea de si ir a Zerind o a Sibiu es un mejor primer paso.



Estrategias de búsqueda Ciega

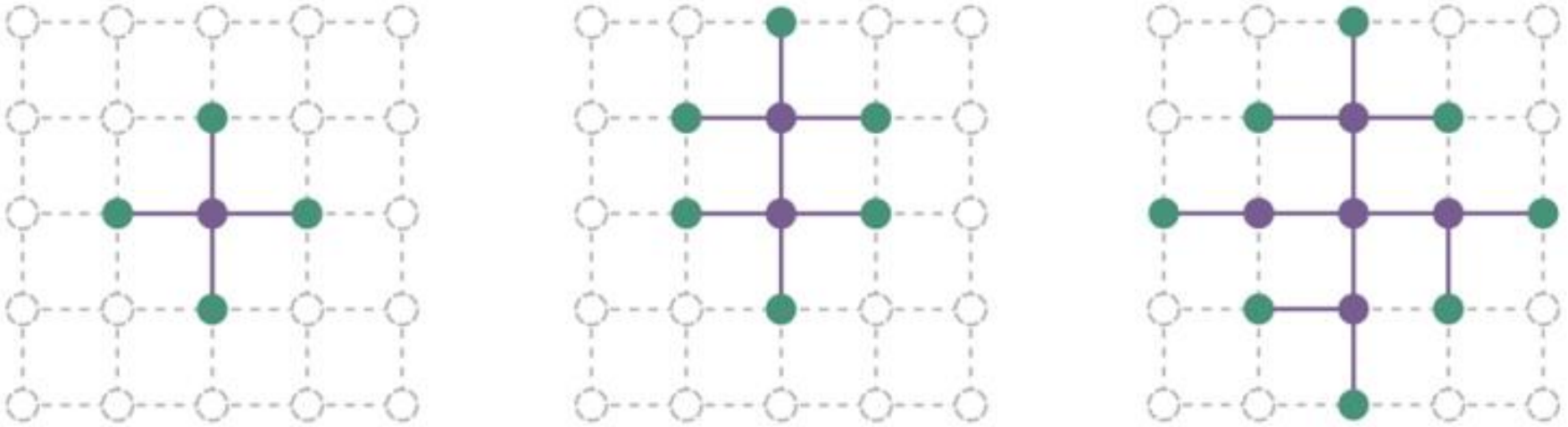
UNIVERSIDAD EAFIT | Escuela de Ciencias Aplicadas e Ingeniería

Nosotros 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.



Estrategias de búsqueda Ciega

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 (y los estados correspondientes) que han sido expandidos; y el exterior es el conjunto de estados que no han sido alcanzados.



Estrategias de búsqueda Ciega

Recuerda… 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. Un nodo en el árbol se representa mediante una estructura de datos con cuatro componentes:

- **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

Tipos de colas se utilizan en los algoritmos de búsqueda:

- **Priority queue**: Este tipo de cola extrae primero el nodo con el costo mínimo de acuerdo con alguna función de evaluación. Se utiliza en la búsqueda del mejor primero (*best-first search*).

- **FIFO queue (First-In-First-Out)**, o cola de primero en entrar, primero en salir, extrae primero el nodo que se agregó a la cola en primer lugar; veremos que se utiliza en la búsqueda en anchura (*breadth-first search*).

- **LIFO queue (Last-In-First-Out)**, también conocida como pila (*stack*), extrae primero el nodo que se agregó más recientemente; veremos que se utiliza en la búsqueda en profundidad (*depth-first search*).

Estrategias de búsqueda Ciega- Uniformed Search Algorithm

UNIVERSIDAD EAFIT | Escuela de Ciencias Aplicadas e Ingeniería

Uniformed Search Algorithm:
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 el **algoritmo de Dijkstra** en la comunidad de ciencias de la computación teórica, y como **búsqueda de costo uniforme** en la comunidad de IA.

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

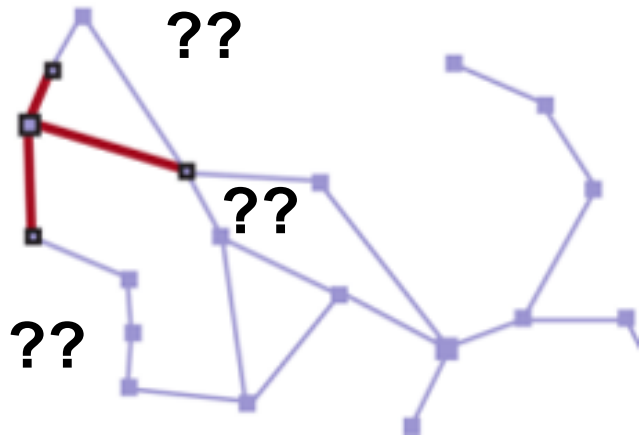
Estrategias de búsqueda Ciega- Uniformed Search Algorithm

UNIVERSIDAD EAFIT | Escuela de Ciencias Aplicadas e Ingeniería

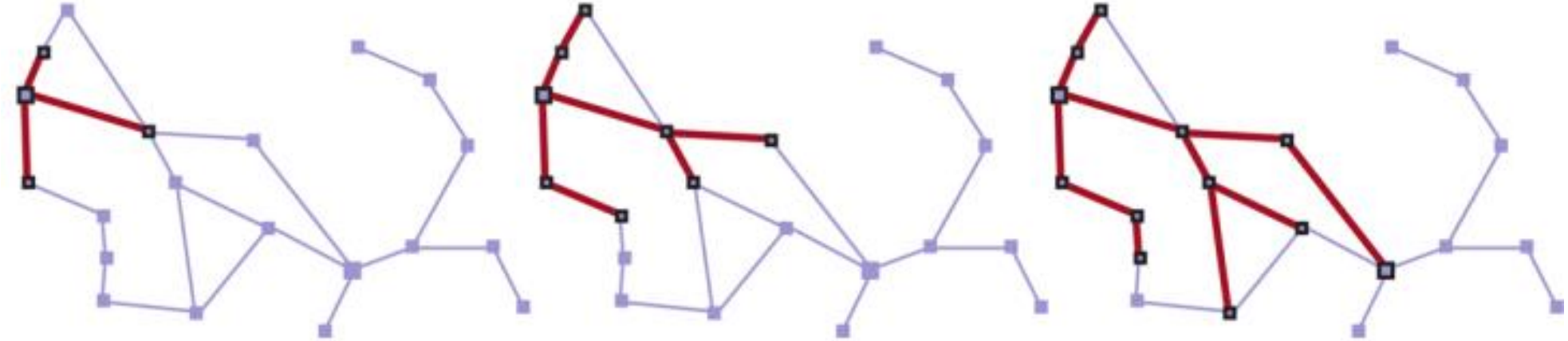
```
function BEST-FIRST-SEARCH(problem, f) returns a solution node or failure
  node ← NODE(STATE=problem.INITIAL)
  frontier ← a priority queue ordered by f, with node as an element
  reached ← a lookup table, with one entry with key problem.INITIAL and value node
  while not IS-EMPTY(frontier) do
    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 un objetivo..

Estrategias de búsqueda Ciega- Uniformed Search Algorithm



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. Cada nodo hijo se agrega a la frontera si no ha sido alcanzado antes, o se vuelve a agregar si ahora se alcanza con un costo de camino menor que cualquier camino anterior.



Estrategias de búsqueda Ciega

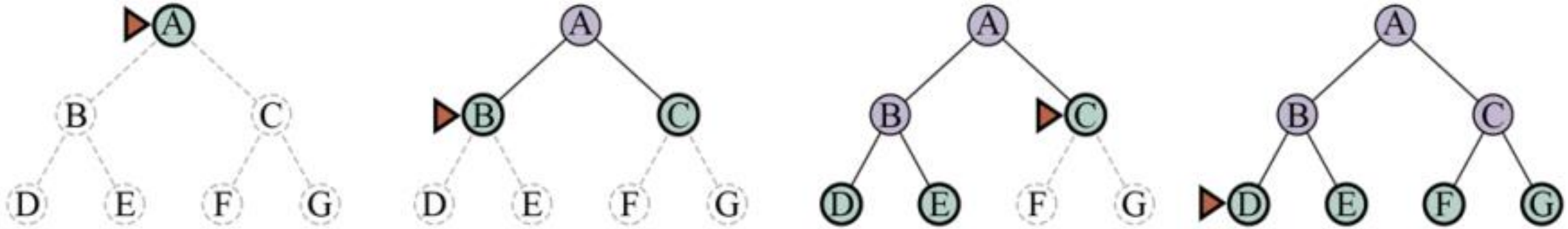
UNIVERSIDAD EAFIT
Escuela de Ciencias Aplicadas e Ingeniería

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

UNIVERSIDAD EAFIT | Escuela de Ciencias Aplicadas e Ingeniería

NOTA1: En este caso ‘**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, verificando si un nodo es una solución tan pronto como se genera.

NOTA2: 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.

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
```


Estrategias de búsqueda Ciega

UNIVERSIDAD EAFIT
Escuela de Ciencias Aplicadas e Ingeniería

CODE AND CLASS EXERCISE

Tareas

- Teniendo en cuenta las bases vistas en clase, investigar en qué consisten las técnicas de Depth First Search e Iterative Deepening search

- Estudiar el Notebook de Iterative Deepening search suministrado por el profesor

- Investigar en qué consisten las métricas de rendimiento de los algoritmos de búsqueda y como pueden ser estimadas: completeness,cost optimality,time complexity,space complexity

Referencias

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