# INTELIGENCIA ARTIFICIAL (1INF24)

## UNIDAD 1: Introducción a la IA. Búsqueda y optimización en IA

*Tema 2: La Búsqueda dentro de la Inteligencia Artificial*
(Parte 1)

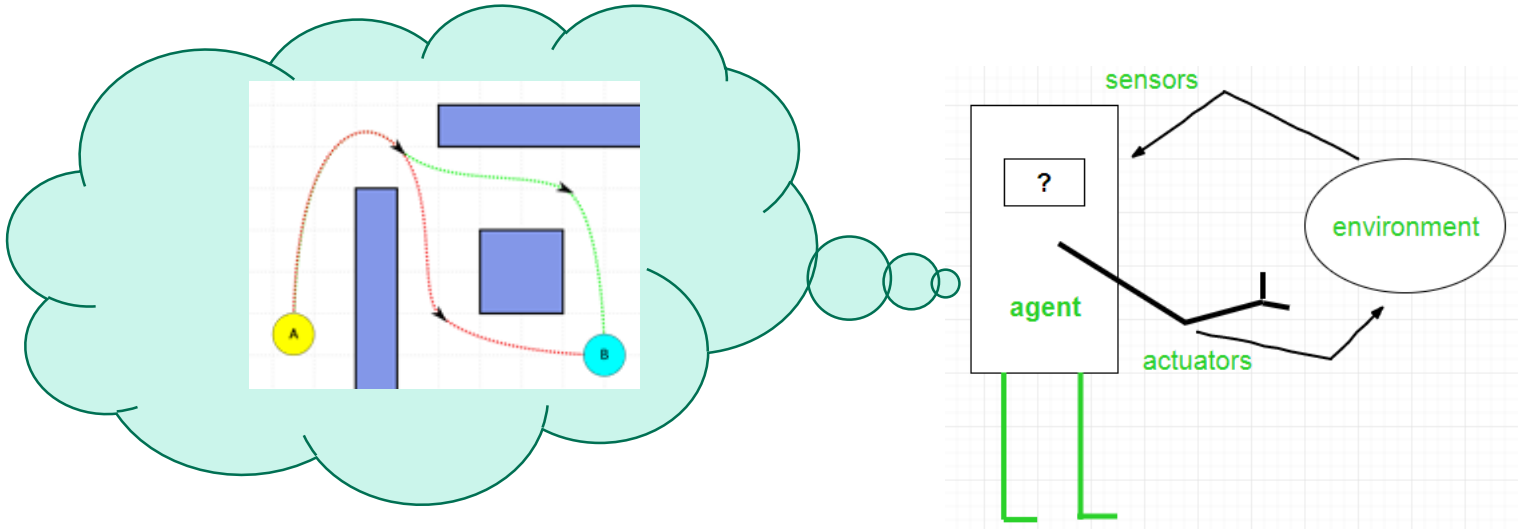
Dr. Edwin Villanueva Talavera

# Contenido

- Agentes de Resolución de Problemas
- Búsqueda de Soluciones
- Búsqueda Sin Información

# Agentes de resolución de problemas

- En muchos problemas se necesita encontrar una secuencia de acciones (ruta o camino) para llegar a un estado objetivo
- En ese caso podemos construir un tipo de agente basado en objetivo llamado de agente de resolución de problemas
- Este tipo de agente usa representación atómica de los estados



# Agentes de resolución de problemas

Pasos de un agente básico de resolución de problemas:

- Formulación de objetivo
- Formulación de problema:
  - Estado inicial, espacio de estados, acciones, modelo de transición, costo de camino
- Búsqueda de solución:
  - Encuentra una secuencia de acciones para llegar a un estado objetivo
- Ejecución de solución

# Agentes de resolución de problemas

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

La suposición es un ambiente conocido, estático, observable, discreto y determinístico.

# Agentes de resolución de problemas

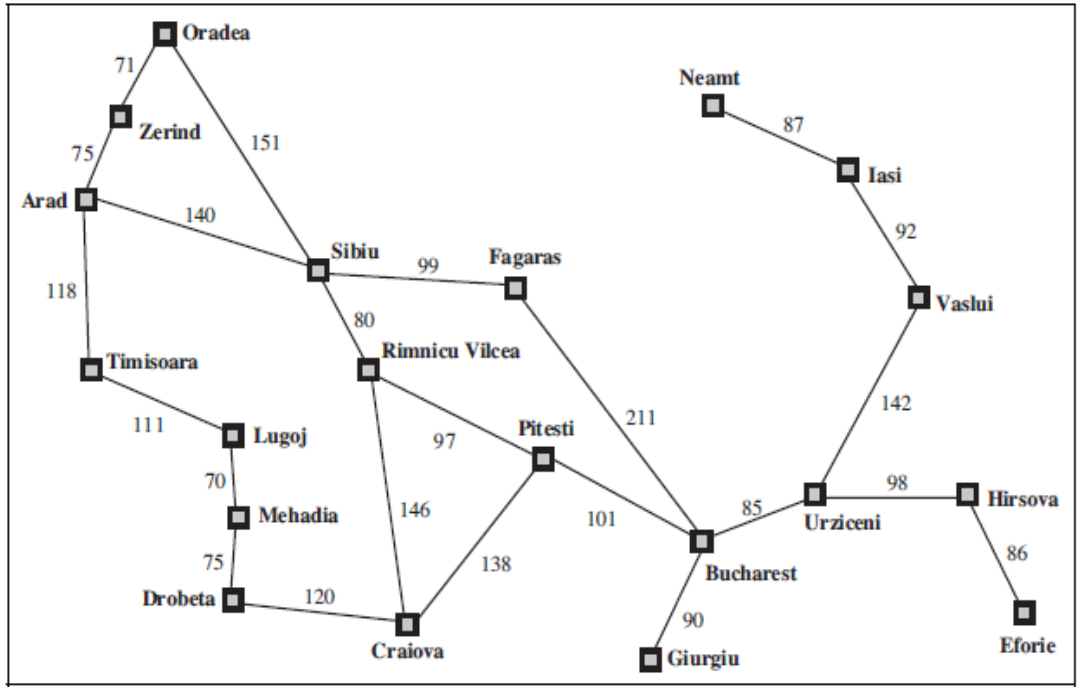
Componentes de la Formulación del Problema:

| | |
|---|---|
| Estados: | • Conjunto de situaciones diferentes que puede estar el problema |
| Estado inicial: | • Situación inicial del problema |
| Acciones: | • Operaciones que pueden ser realizadas desde un determinado estado *s*, denotada comúnmente como: ACTIONS(*s*) |
| Modelo de transición: | • Estados alcanzables desde un estado dado *s* con una determinada acción *a*, comúnmente se denota: RESULT(*s*,*a*) |
| Función de prueba de objetivo: | • Determina si un estado es la solución del problema, comúnmente se denota: GOAL-TEST(*s*) |
| Costo del camino: | • Alguna función que mide cuan difícil o costoso es determinado camino para llegar a un nodo *s* desde el estado inicial, *g*(*s*) |

# Agentes de resolución de problemas

Ejemplo de Formulación de Problema: búsqueda de ruta en mapa

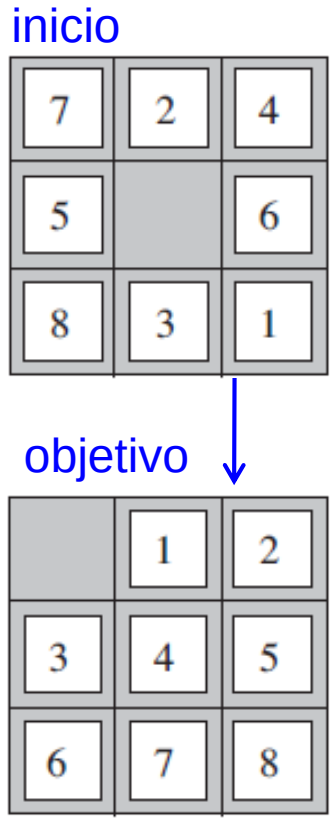
- Estados: Todas las posibles ciudades
- Estado inicial: Ciudad inicial
- Acciones: Moverse a alguna ciudad vecina
- Modelo de transición: Mapa
- Prueba de Objetivo: Verificar si se llego a la ciudad deseada
- Costo del camino: Puede ser tiempo, distancia recorrida, contaminación emitida, etc.



# Agentes de resolución de problemas

Ejemplo de Formulación de Problema: 8-puzzle

- Estados: Todas las configuraciones posibles de 8 números y un blanco
- Estado inicial: Alguna configuración dada del puzzle
- Acciones: Movimientos del casillero blanco: **Derecha, Izquierda, Arriba, Abajo**
- Modelo de transición: resultado de alguna acción, dado un estado:
  - Ej. RESULT(inicio, Izquierda) = blanco y 5 intercambiados
- Prueba de Objetivo: Verifica si el estado es el objetivo
- Costo del camino: Cada acción cuesta 1. El costo de la solución seria el costo de todas las acciones



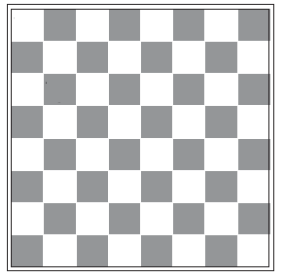
# Agentes de resolución de problemas

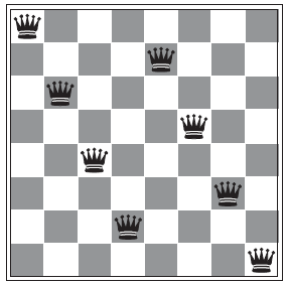
Ejemplo de Formulación de Problema: 8-queens

- Estados: configuraciones de 0 a 8 reinas en el tablero
- Estado inicial: 0 reinas en el tablero
- Acciones: **Adicionar** una reina a un casillero vacío
- Modelo de transición: Retorna el tablero con la reina añadida
- Prueba de objetivo: Verificar que el estado tenga 8 reinas no atacadas

Esta formulación tiene
64x63x…x57 ~**$1.8 \times 10^{14}$** posibles secuencias a investigar!

inicio





Objetivo:
estado donde las 8 reinas no se ataquen

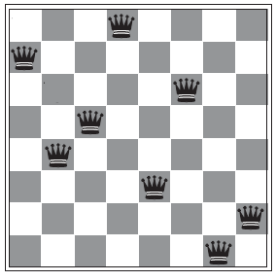
# Agentes de resolución de problemas

Ejemplo de otra formulación de Problema: 8-queens

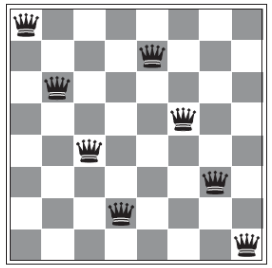
- Estados: Vectores de 8 números no repetidos (permutaciones). Cada elemento indica la fila en que se encuentra la reina en una columna
- Estado inicial: Permutación aleatoria
- Acciones: Intercambiar 2 elementos
- Prueba de objetivo: Verificar si la nueva permutación tiene reinas no atacadas

Esta formulación tiene
8x7x…x1 = **40320** posibles secuencias a investigar!

inicio



[2, 5, 4, 1, 6, 3, 8, 7]



[1, 3, 5, 7, 2, 4, 6, 8]

Objetivo:
estado donde las 8 reinas no se ataquen

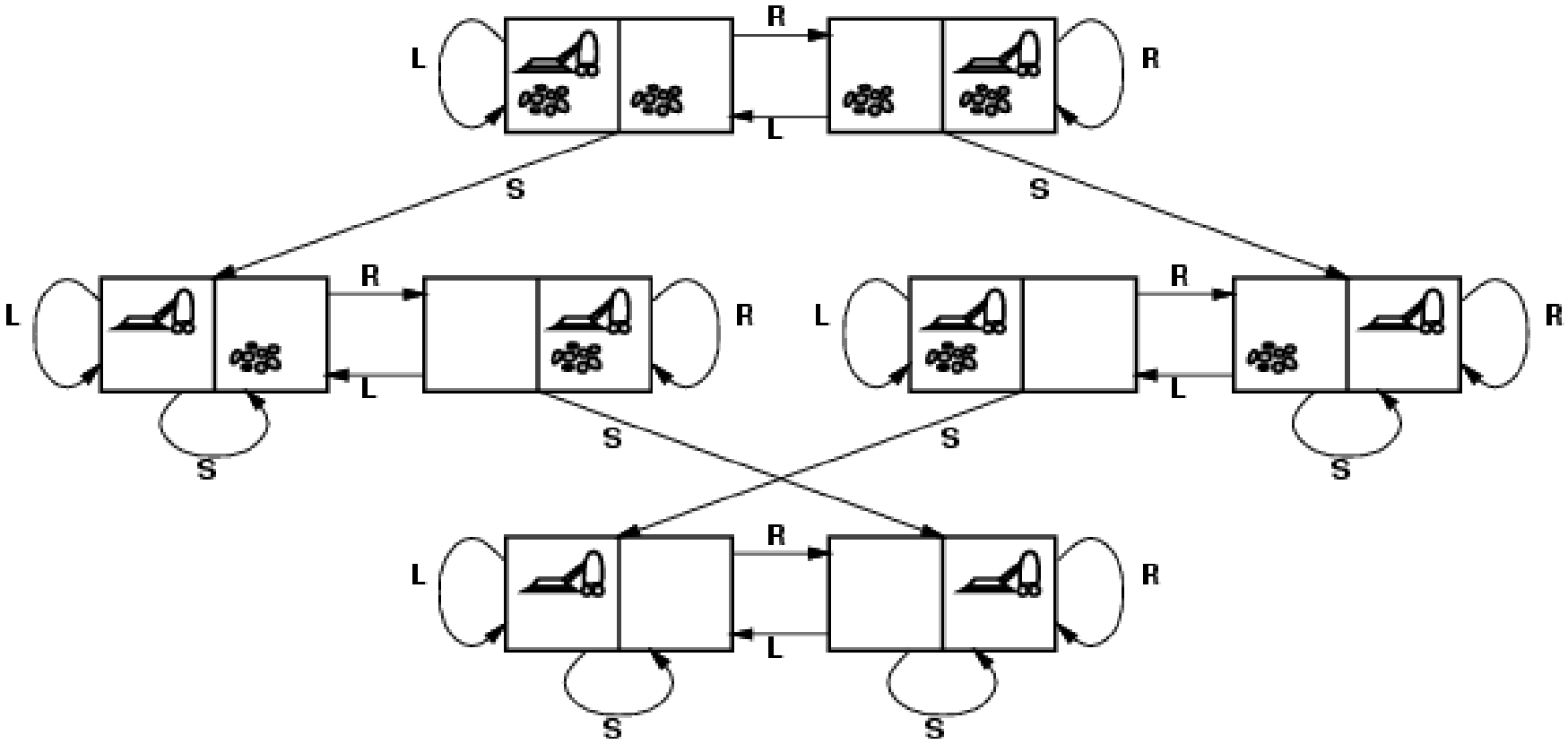
# Agentes de resolución de problemas

## Espacio de estados

- Conjunto de todos los estados accesibles a partir de un estado inicial
  - El estado inicial, las acciones y el modelo de transición (o **función sucesor**) determinan el espacio de estados
- El espacio de estados puede ser representado con un grafo, donde los nodos representan estados y los arcos acciones

# Agentes de resolución de problemas

## Ejemplo de espacio de estados del mundo de la aspiradora

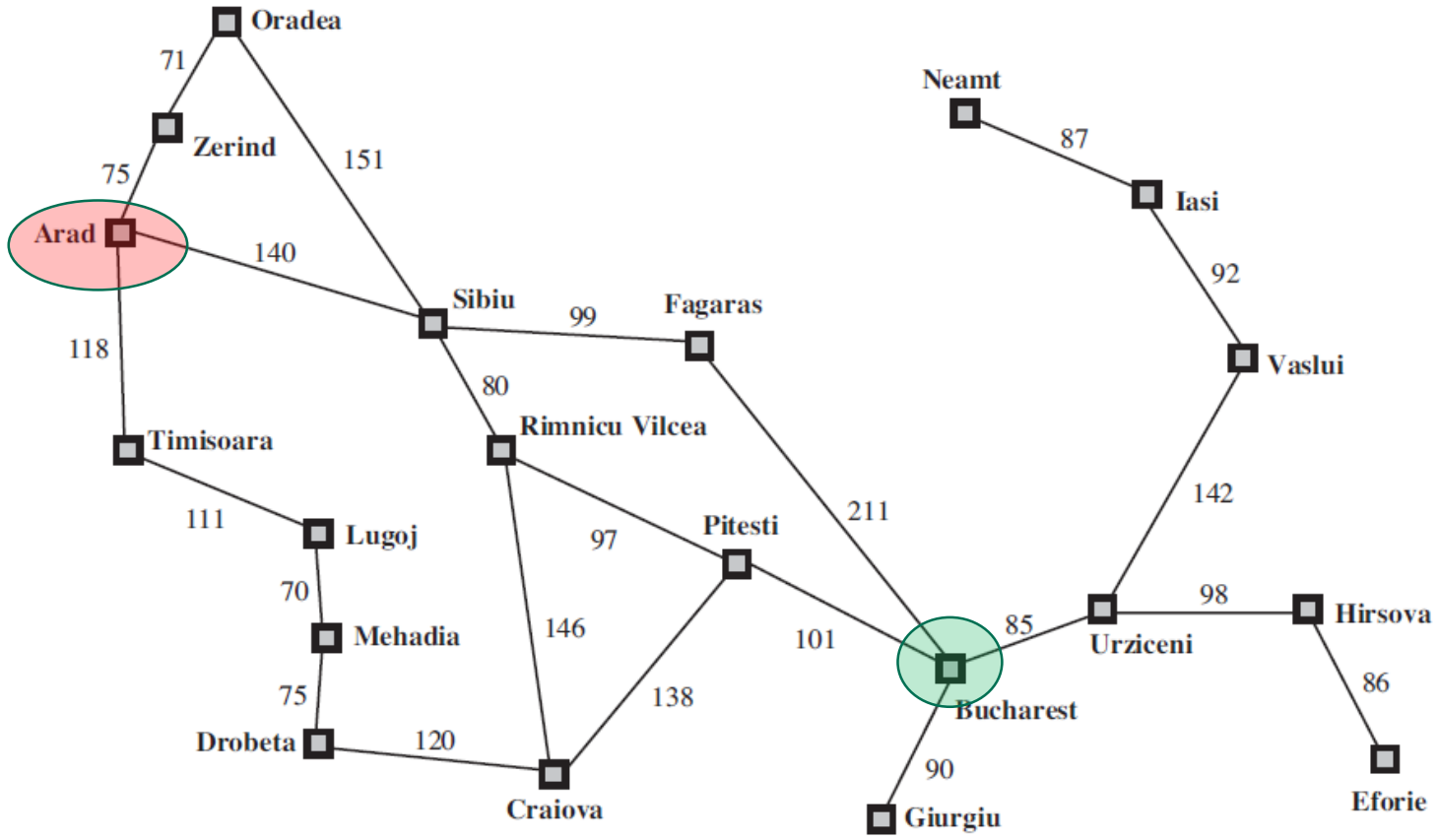


# Búsqueda de Soluciones

- La idea es explorar el espacio de estados mediante el recorrido de un árbol de búsqueda
- Expandir el estado actual aplicando la función sucesor, generando nuevos estados
- La estrategia de búsqueda determina el camino a seguir, esto es, que nodos se exploran primero y cuáles se dejan para después.

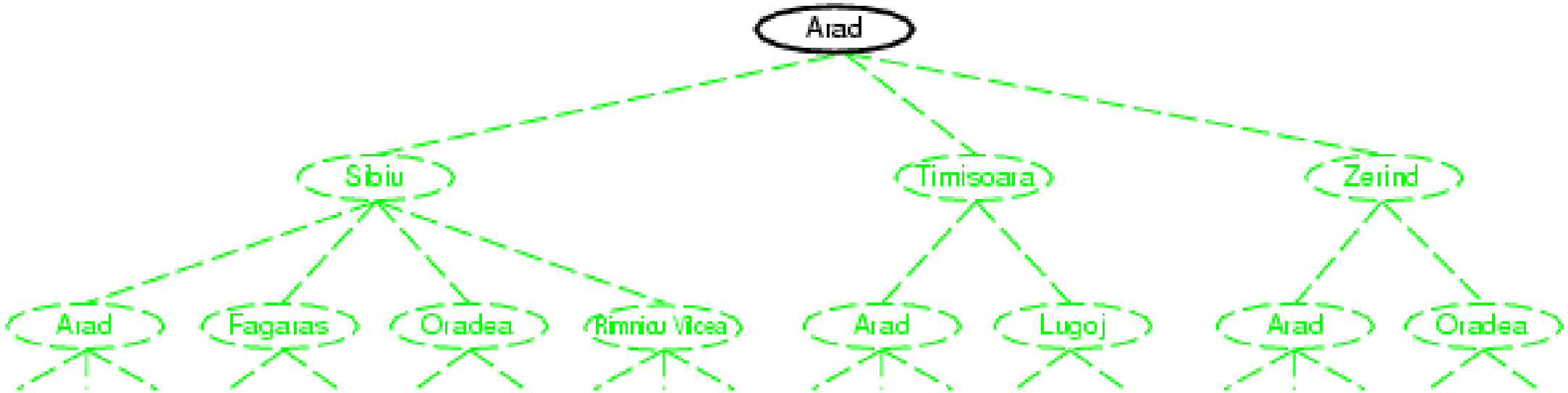
# Búsqueda de Soluciones

## Ejemplo: Búsqueda en el mapa de Romania



# Búsqueda de Soluciones

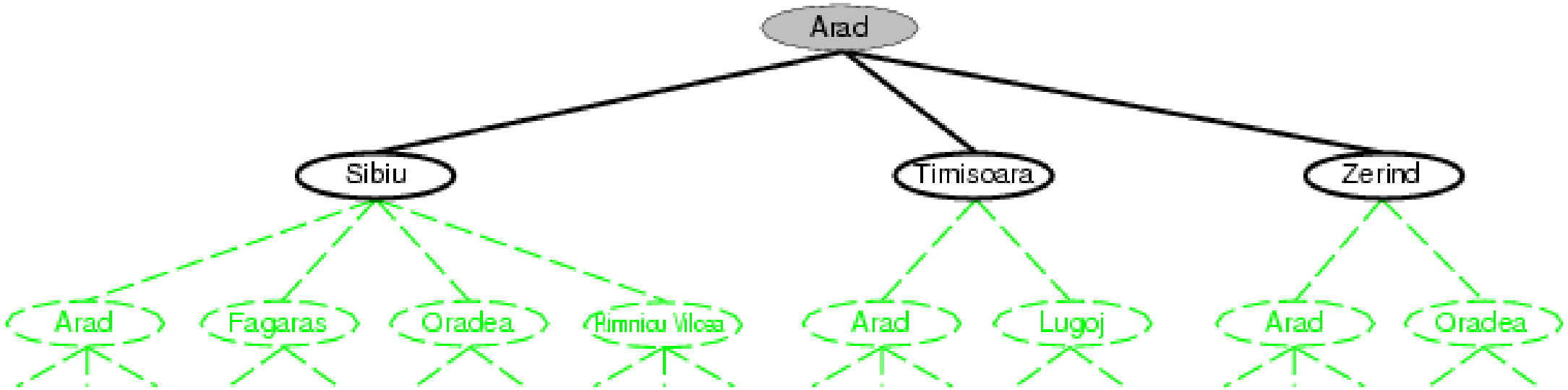
Ejemplo: Búsqueda en el mapa de Romania




Estado Inicial

# Búsqueda de Soluciones

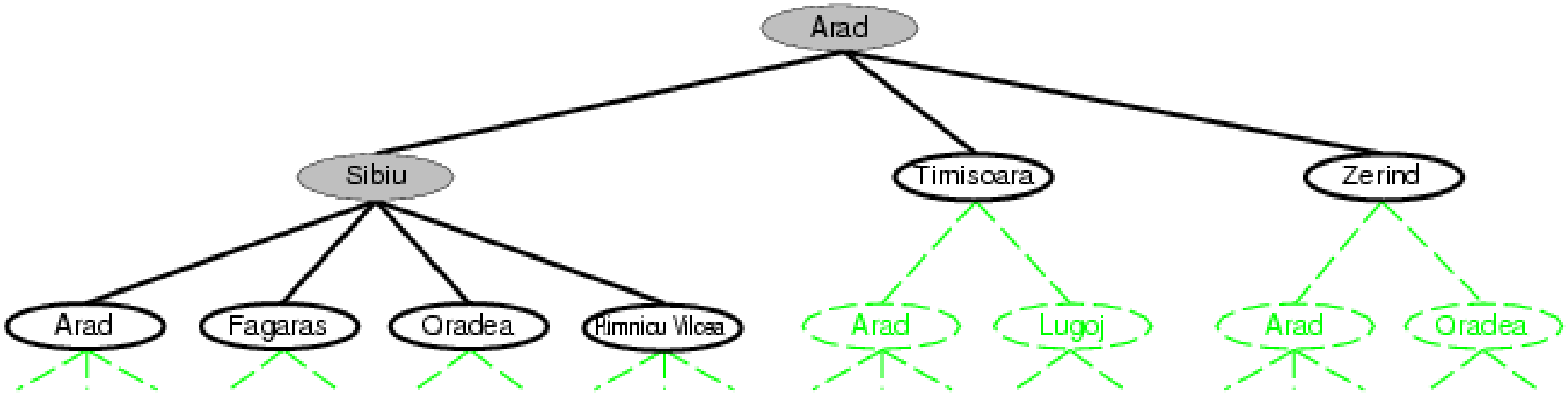
## Ejemplo: Búsqueda en el mapa de Romania




Después de expandir Arad

# Búsqueda de Soluciones

## Ejemplo: Búsqueda en el mapa de Romania




Después de expandir Sibiu

# Búsqueda de Soluciones

Árbol de búsqueda ≠ a espacio de estados!

- Un Nodo del árbol es una estructura de datos que implementa el árbol de búsqueda. Un estado es una configuración física.
  - Por ejemplo, el mapa de Romania tiene 20 estados, mientras que el árbol de búsqueda de Romania tiene tamaño infinito, ya que hay infinitos caminos del tipo:

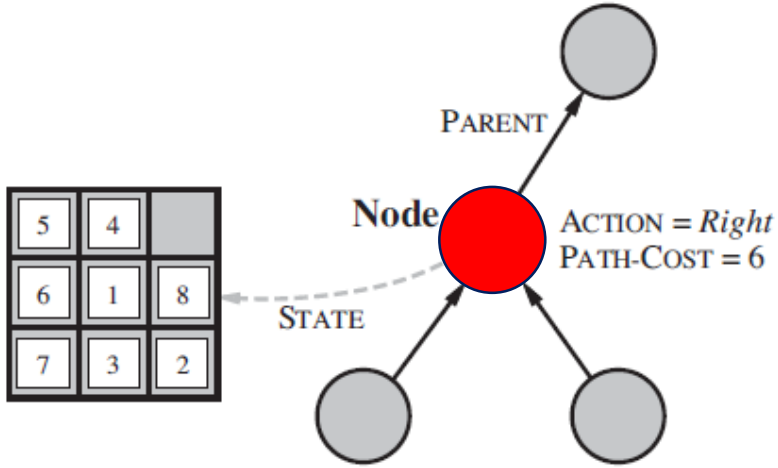
    Arad-Sibiu-Arad-Sibiu-Arad-...

- En el grafo de espacio de estados cada estado es representado por un único nodo.
- En un árbol de búsqueda varios nodos pueden representar un mismo estado (cuando hay varios caminos hacia ese estado).

# Búsqueda de Soluciones

## Estructura de un Nodo:

- Debe incluir información de: estado, nodo padre, la acción que generó el nodo, costo del camino desde el nodo raíz, y profundidad del nodo




- La colección de nodos que fueron generados pero aún no expandidos es llamada de **frontera**
- La forma como colocar/sacar nodos de la frontera define la **estrategia de búsqueda**

# Búsqueda de Soluciones

Generación de nodos hijos:

```
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)
```

# Búsqueda de Soluciones

Estructuras de datos para implementar la frontera: ***queue***

- First-in First Out (FIFO)
- Last-in First-out (LIFO o Pila)
- Cola de Prioridad

Operaciones en la frontera:

- *EMPTY?(queue)*: Retorna *true* si la cola esta vacía
- *POP(queue)*: Remueve y retorna el 1er elemento de la cola
- *INSERT(element, queue)*: Inserta un elemento en la cola y devuelve esta

# Búsqueda sin Información

## Búsqueda sin información o búsqueda ciega

- Estrategias de búsqueda sin información usan solamente la información disponible en la definición del problema
  - Solo generan sucesores verificando si es estado objetivo
- Las estrategias de búsqueda sin información se distinguen por la orden en que los nodos son expandidos.
  - Búsqueda en amplitud (*Breadth-first search*)
  - Búsqueda de costo uniforme
  - Búsqueda en profundidad (*Depth-first search*)
  - Búsqueda en profundidad limitada
  - Búsqueda de profundización iterativa
  - Búsqueda bidireccional

# Búsqueda sin Información

Estrategia general de **búsqueda sin memoria de estados visitados** (búsqueda en árbol)

```
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
```

# Búsqueda sin Información

Estrategia general de **búsqueda con memoria de estados visitados** (búsqueda en grafo)

```
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
```

# Búsqueda sin Información

## Evaluación de desempeño

- Estrategias son evaluadas de acuerdo a los siguientes criterios
  - Completitud: el algoritmo siempre encuentra la solución?
  - Complejidad de tiempo: número de nodos generados
  - Complejidad de espacio: número máximo de nodos en memoria
  - Optimalidad: la estrategia encuentra la solución óptima?
    - Una solución óptima es una solución con menor costo de camino.
- Complejidad de tiempo y espacio son medidos en función de:
  - *b*: máximo factor de ramificación del árbol (número máximo de sucesores de cualquier nodo)
  - *d*: profundidad del nodo objetivo menos profundo
  - *m:* tamaño máximo de cualquier camino en el espacio de estados

# Búsqueda sin Información

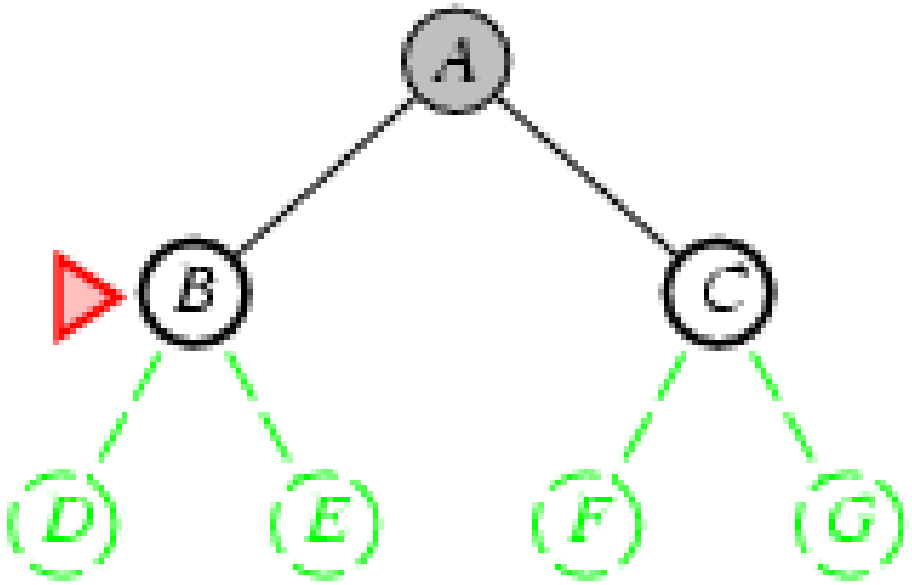
## Búsqueda en amplitud (*Breadth-first search*)

- Expandir el nodo aun no expandido mas cerca de la raíz
- Implementación: Puede ser TREE-SEARCH o GRAPH-SEARCH usando como frontera una cola FIFO:

```
function BREADTH-FIRST-SEARCH(problem) returns a solution, or failure
  node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
  frontier ← a FIFO queue with node as the only element
  explored ← an empty set
  loop do
    if EMPTY?(frontier) then return failure
    node ← POP(frontier)  /* chooses the shallowest node in frontier */
    add node.STATE to explored
    for each action in problem.ACTIONS(node.STATE) do
      child ← CHILD-NODE(problem, node, action)
      if child.STATE is not in explored or frontier then
        if problem.GOAL-TEST(child.STATE) then return SOLUTION(child)
        frontier ← INSERT(child, frontier)
```

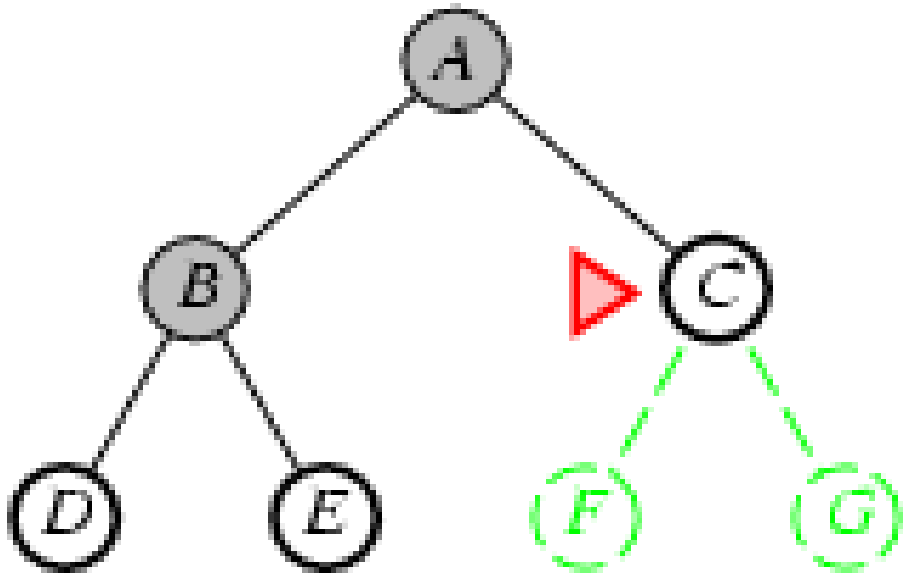
# Búsqueda sin Información

## Búsqueda en Amplitud: ejemplo de exploración de nodos



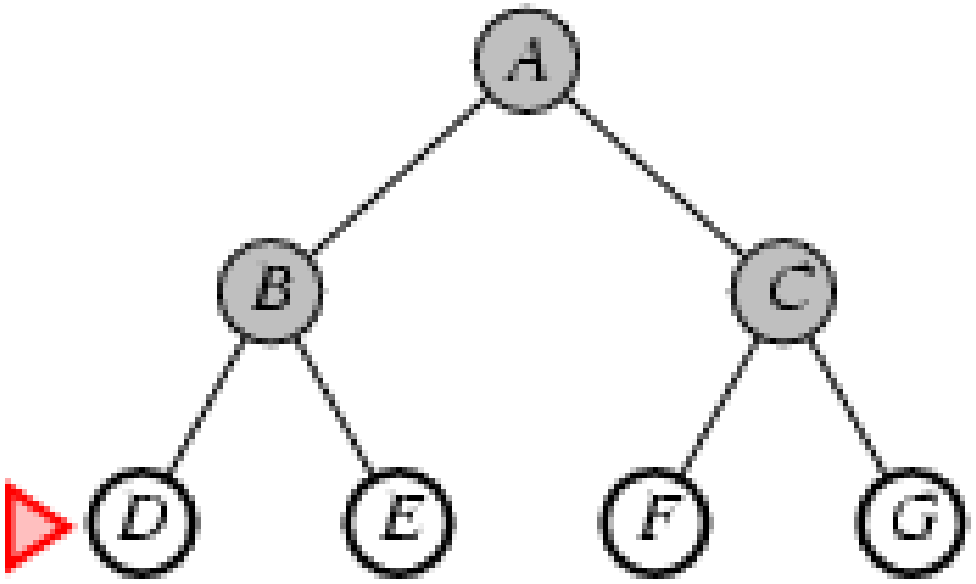
# Búsqueda sin Información

## Búsqueda en Amplitud: ejemplo de exploración de nodos



# Búsqueda sin Información

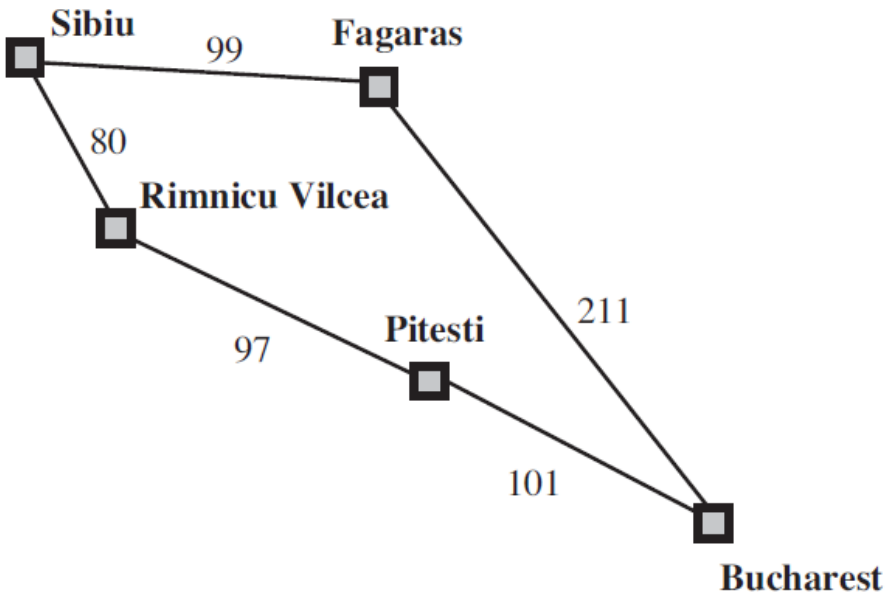
## Búsqueda en Amplitud: ejemplo de exploración de nodos



# Búsqueda sin Información

## Búsqueda en amplitud (Ejercicio)

- Aplicar búsqueda en amplitud en el mapa de Rumania para llegar a *Bucharest* partiendo de *Sibiu*



# Búsqueda sin Información

## Propiedades de Búsqueda en amplitud

- Completa? SI, si b es finito
- Complejidad de tiempo:

  $1+b+b^2+b^3+\ldots+b^d = O(b^d)$ (impl. BFS)

  $1+b+b^2+b^3+\ldots+b^d + b(b^d) = O(b^{d+1})$ (impl. Graph-Search)
- Complejidad de espacio:
  - Existe $O(b^{d-1})$ nodos en *explored set* y $O(b^d)$ en la frontera, así que la complejidad espacial es dominada por la frontera: $O(b^d)$
- Optima? SI, si todos las acciones tuvieran los mismos costos

# Búsqueda sin Información

## Propiedades de Búsqueda en amplitud

- Con un factor de ramificación b=10 y suponiendo que puedan ser generados 1 millón de nodos por segundo y que cada nodo requiera 1KB de espacio, se tendría:

| 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 sin Información

## Búsqueda en Profundidad

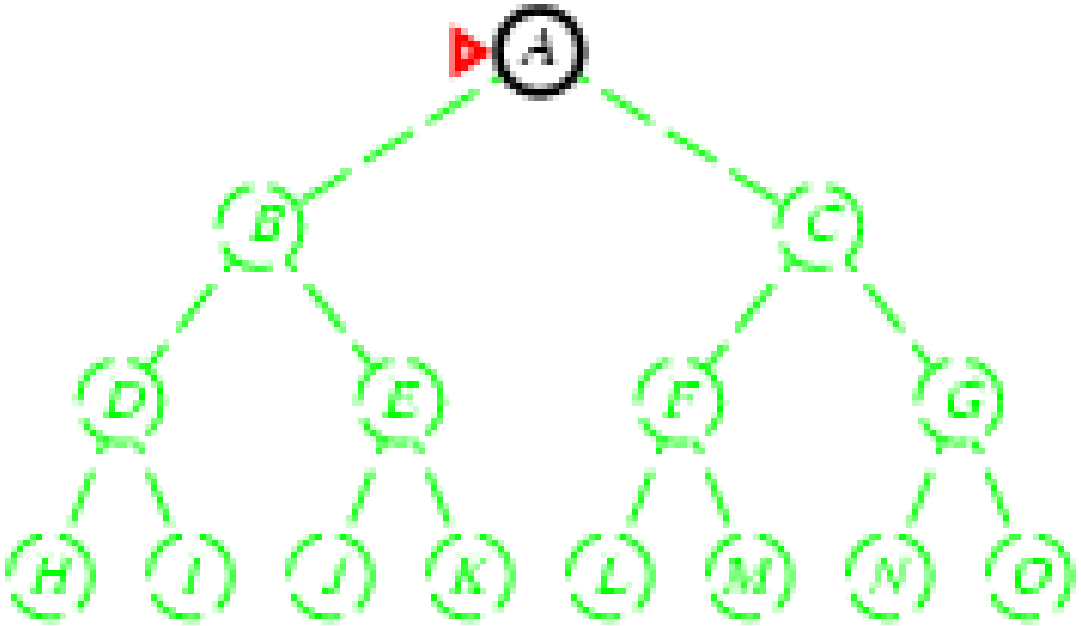
- Expande el nodo no expandido mas profundo
- Implementación: Puede ser GRAPH-SEARCH usando como frontera una lista LIFO (last-in, first-out), también conocida como pila:

```
function DEPTH-FIRST-SEARCH(problem) returns a solution, or failure

  node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  frontier ← a LIFO list (stack) with node as the only element
  explored ← an empty set
  loop do
      if EMPTY?(frontier) then return failure
      node ← POP(frontier)  // chooses the most recent node in frontier
      if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
      add node.STATE to explored
      for each action in problem.ACTIONS(node.STATE) do
          child ← CHILD-NODE(problem, node, action)
          if child.STATE is not in explored or frontier then
              frontier ← INSERT(child, frontier)
```

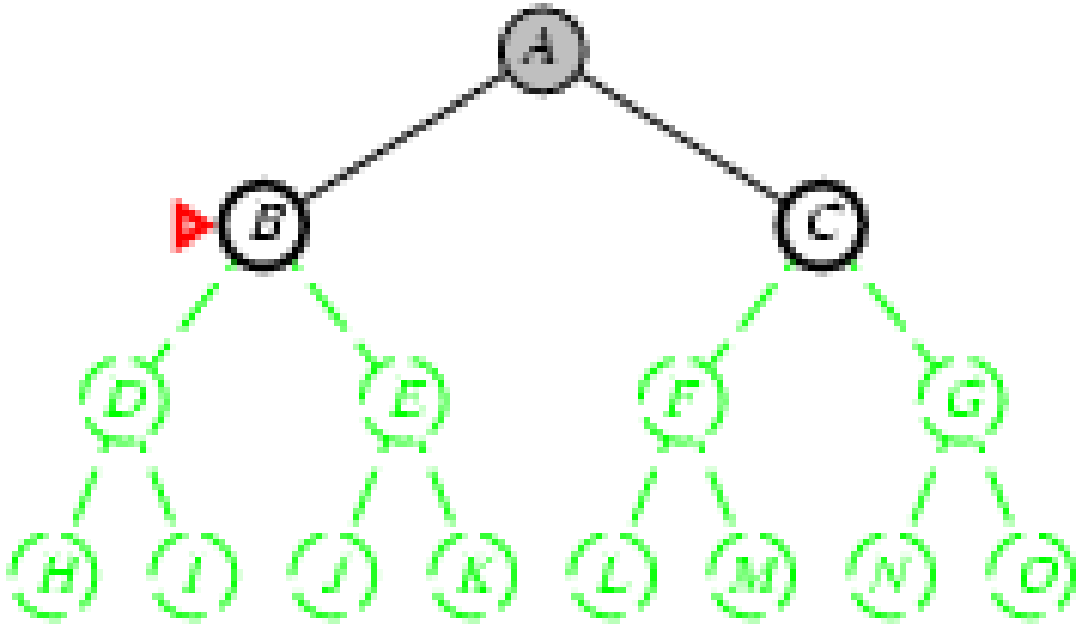
# Búsqueda sin Información

Ejemplo de exploración de nodos en búsqueda en profundidad



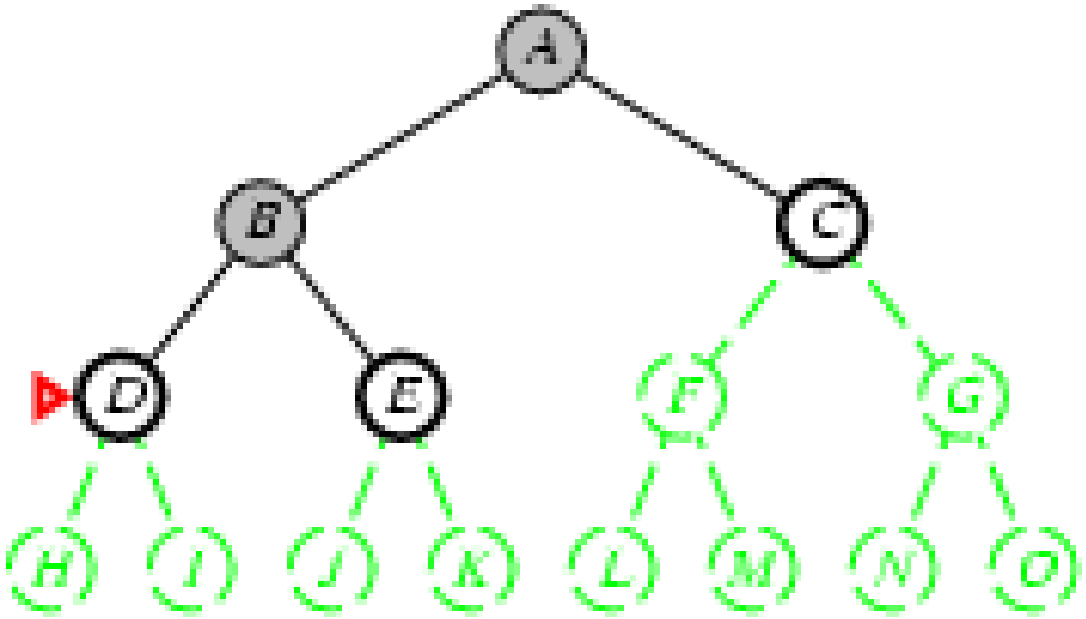
# Búsqueda sin Información

Ejemplo de exploración de nodos en búsqueda en profundidad



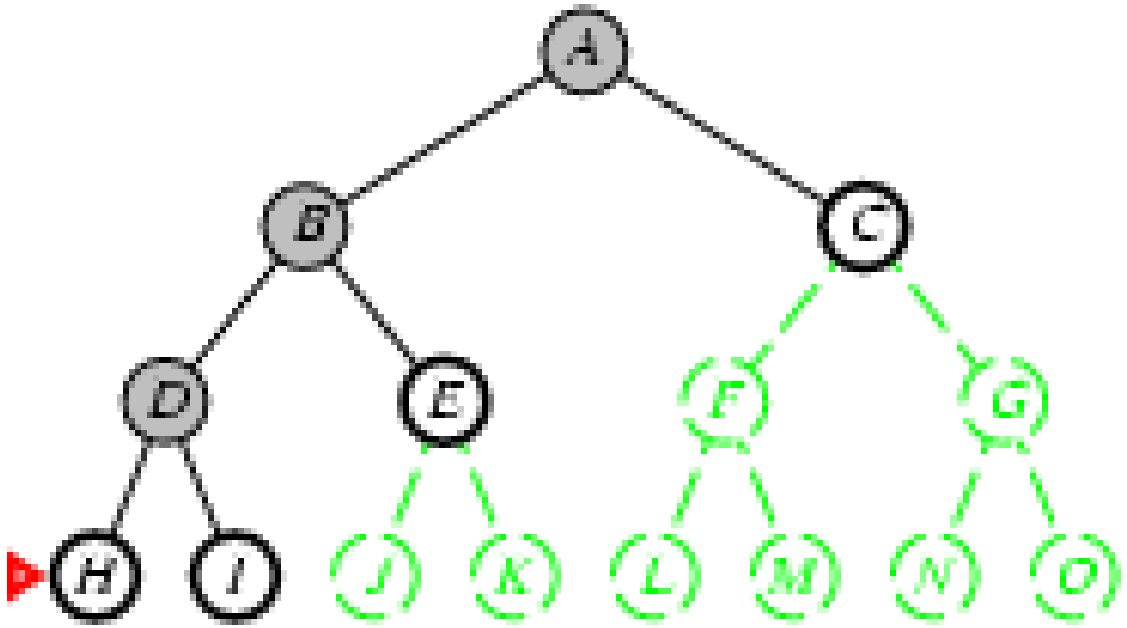
# Búsqueda sin Información

Ejemplo de exploración de nodos en búsqueda en profundidad



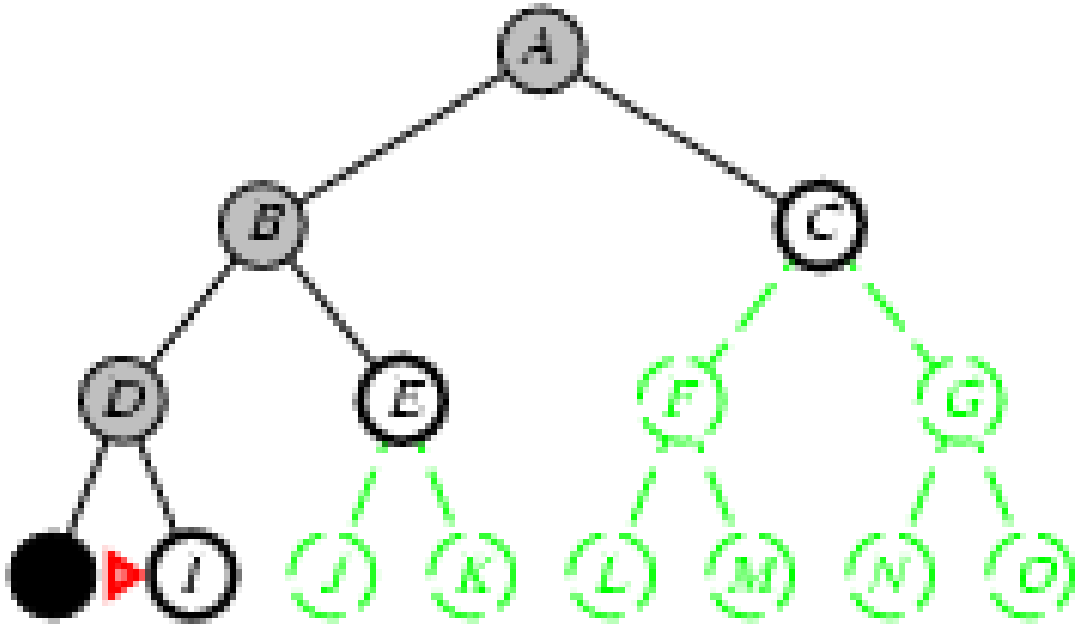
# Búsqueda sin Información

Ejemplo de exploración de nodos en búsqueda en profundidad



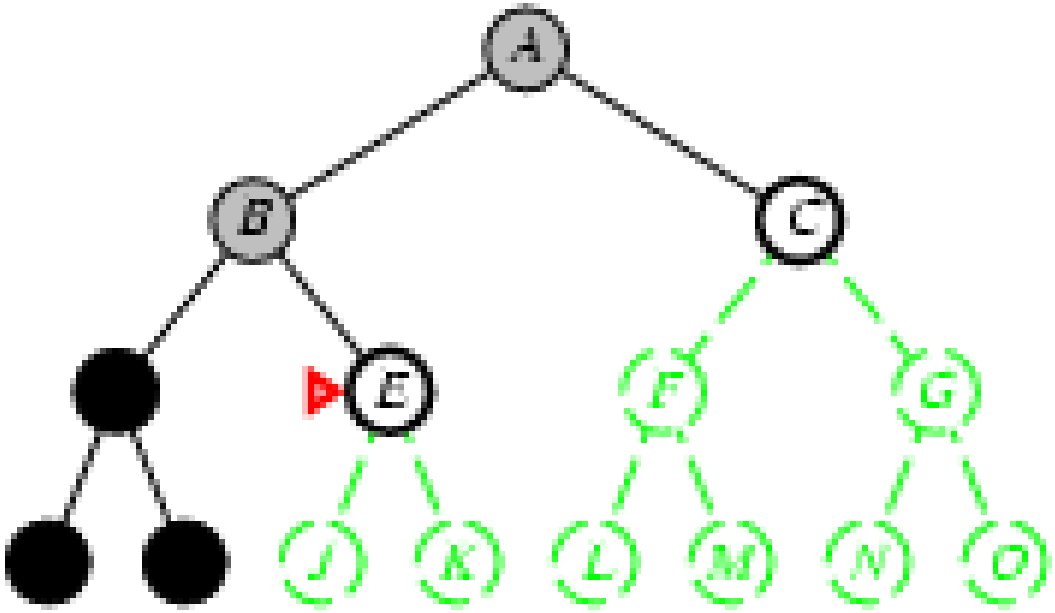
# Búsqueda sin Información

Ejemplo de exploración de nodos en búsqueda en profundidad



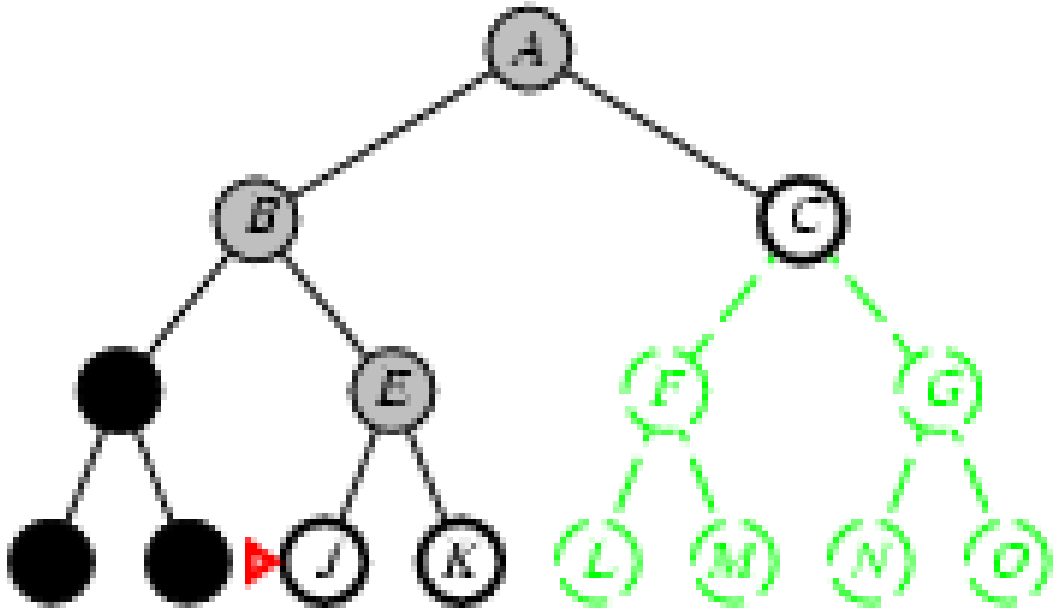
# Búsqueda sin Información

Ejemplo de exploración de nodos en búsqueda en profundidad



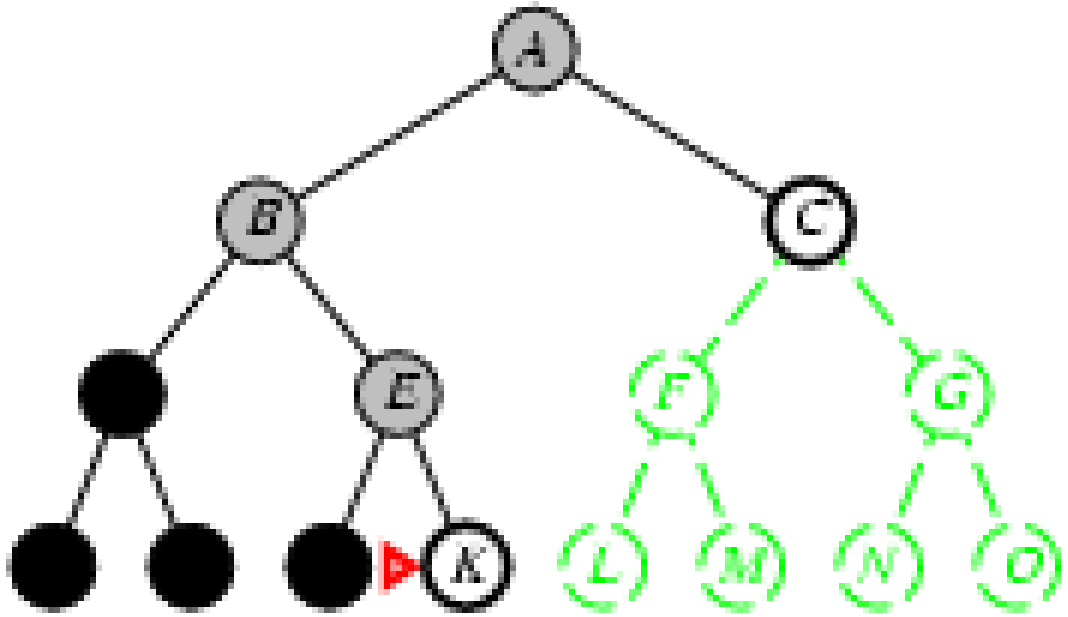
# Búsqueda sin Información

Ejemplo de exploración de nodos en búsqueda en profundidad



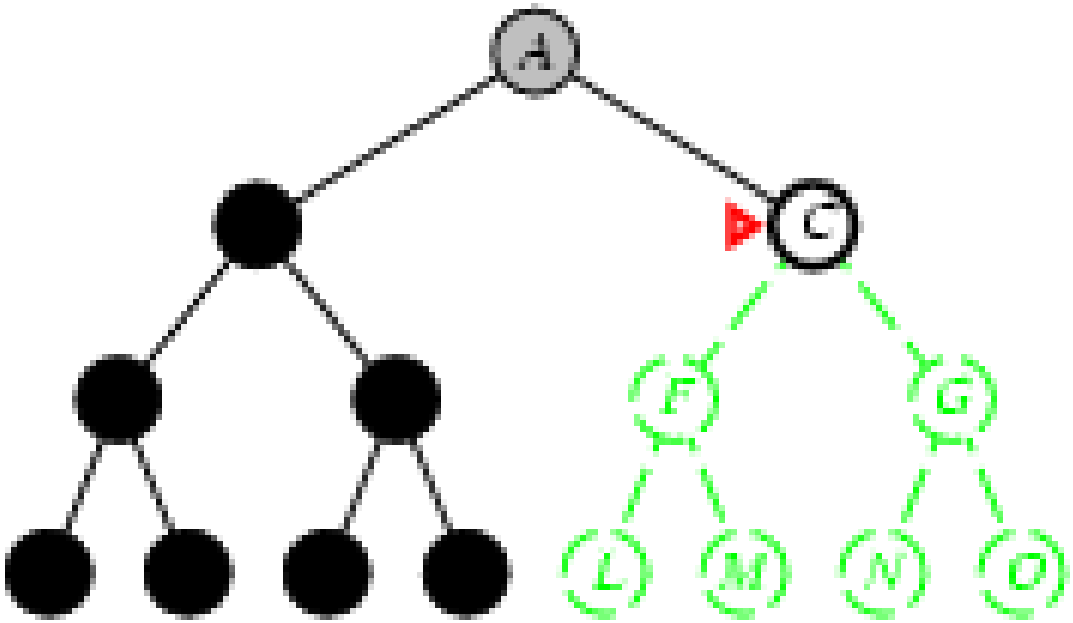
# Búsqueda sin Información

Ejemplo de exploración de nodos en búsqueda en profundidad



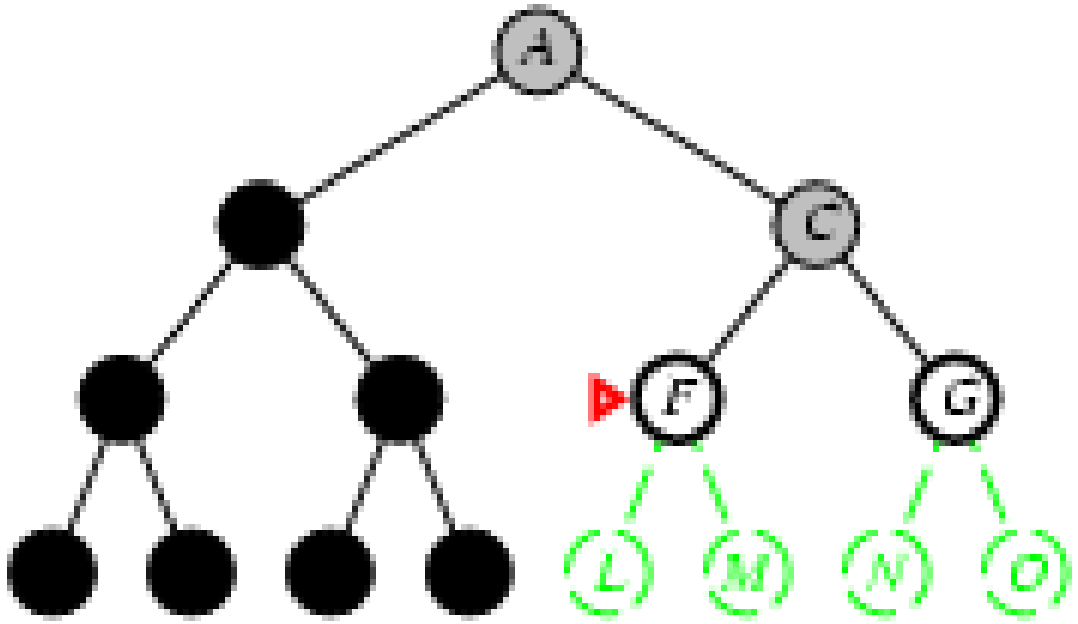
# Búsqueda sin Información

Ejemplo de exploración de nodos en búsqueda en profundidad



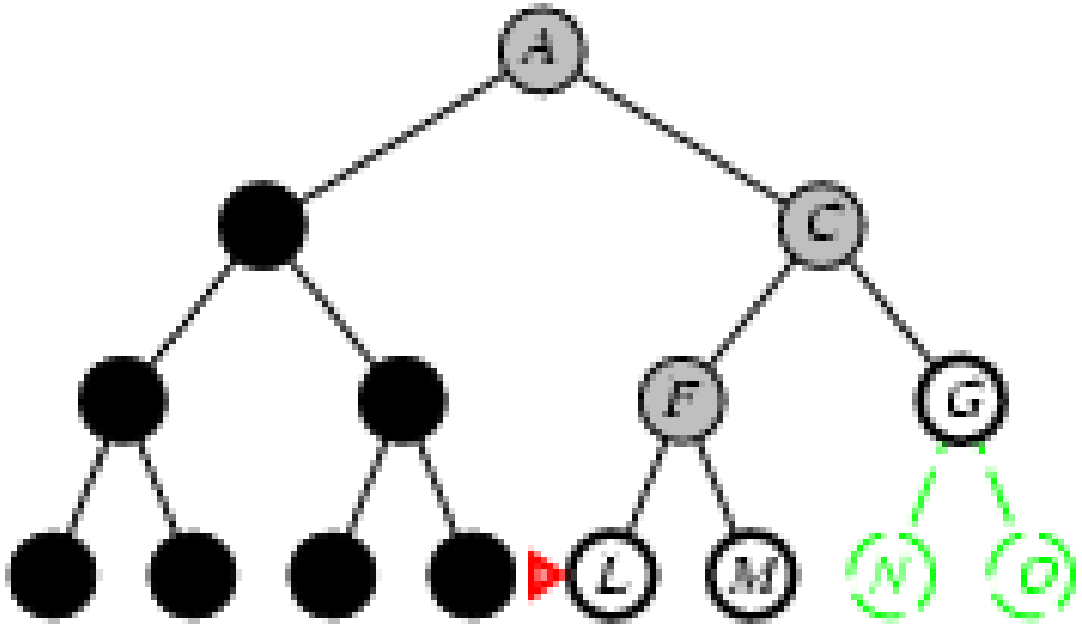
# Búsqueda sin Información

Ejemplo de exploración de nodos en búsqueda en profundidad



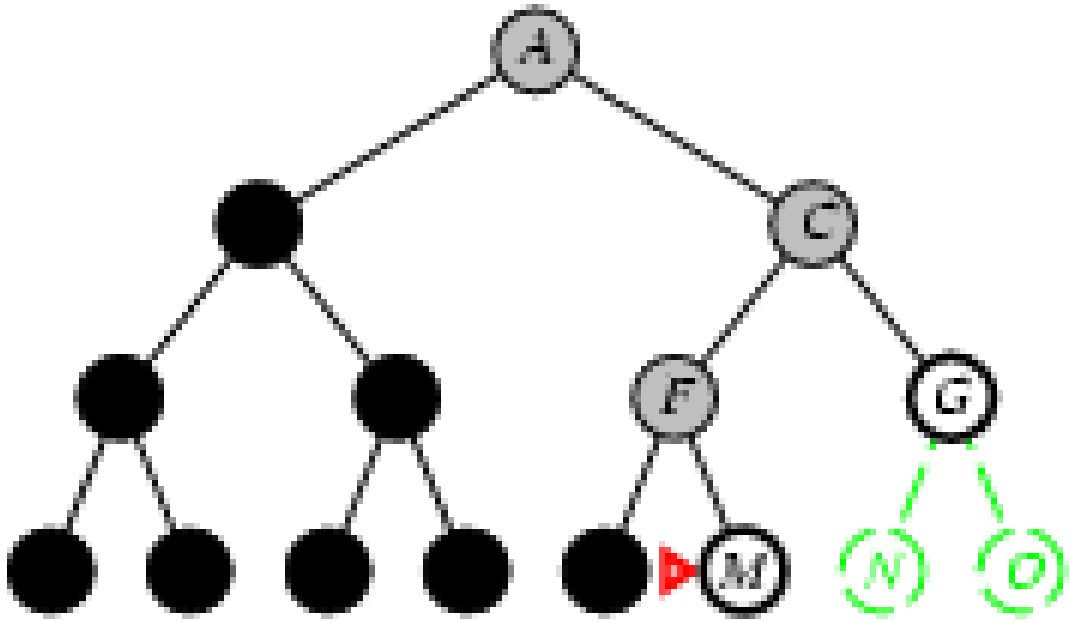
# Búsqueda sin Información

Ejemplo de exploración de nodos en búsqueda en profundidad



# Búsqueda sin Información

Ejemplo de exploración de nodos en búsqueda en profundidad



# Búsqueda sin Información

## Propiedades de Búsqueda en Profundidad

- Completa? SI, solo en espacios con profundidad finita
- Complejidad de tiempo (Implementación TREE-SEARCH):

  $O(b^m)$, pésimo cuando m es mucho mayor que d, pero si hay muchas soluciones puede ser más eficiente que la búsqueda en amplitud
- Complejidad de espacio (Implementación TREE-SEARCH):

  $O(bm)$, (complejidad lineal). En el ejemplo anterior con b=10, d=m=16 se tendría 156 kilobytes en lugar de 10 exabytes
- Optima? NO, ya que la búsqueda termina cuando encuentra la 1ra solución, pudiendo haber otra a una profundidad menor.

# Bibliografía

Capitulo 3.1, 3.2, 3.3 del libro:

Stuart Russell & Peter Norvig “Artificial Intelligence: A modern Approach”, Prentice Hall, Third Edition, 2010

# ¡Gracias!