Inteligencia Artificial

Universidad Tecnológica de la Mixteca

Instituto de Física y Matemáticas
M.C. Alejandro Hernández T.

Capítulo 3

Resolver problemas mediante búsqueda

En donde veremos cómo un agente puede encontrar una secuencia de acciones que alcance sus objetivos, cuando ninguna acción simple lo hará.

Generalidades

- Agente Reactivo (AR): Basa sus acciones en un aplicación directa desde los estados a las acciones, no son buenos en entornos para los cuales tal aplicación sea demasiado grande para almacenarla y que tarde mucho en aprenderla

- Agentes Basados en Objetivos (ABO): Entre los ABO está el agente resolvente-problemas que decide que hacer para encontrar secuencias de acciones que conduzcan a los estados deseables

- Se trabaja con algoritmos **no informados**, en el sentido de que no dan información sobre el problema salvo su definición

Agente de vacaciones

Imagine un agente en la ciudad de Arad en Rumania, disfrutando de unas vacaciones, pero recibe la noticia de que al día siguiente tiene que entrevistarse con alguien en Bucarest. ¿Qué hacer?

Para solucionar un problema el primer paso es la:

Formulación del objetivo: estar en Bucarest

basándose en la situación actual y la medida de rendimiento

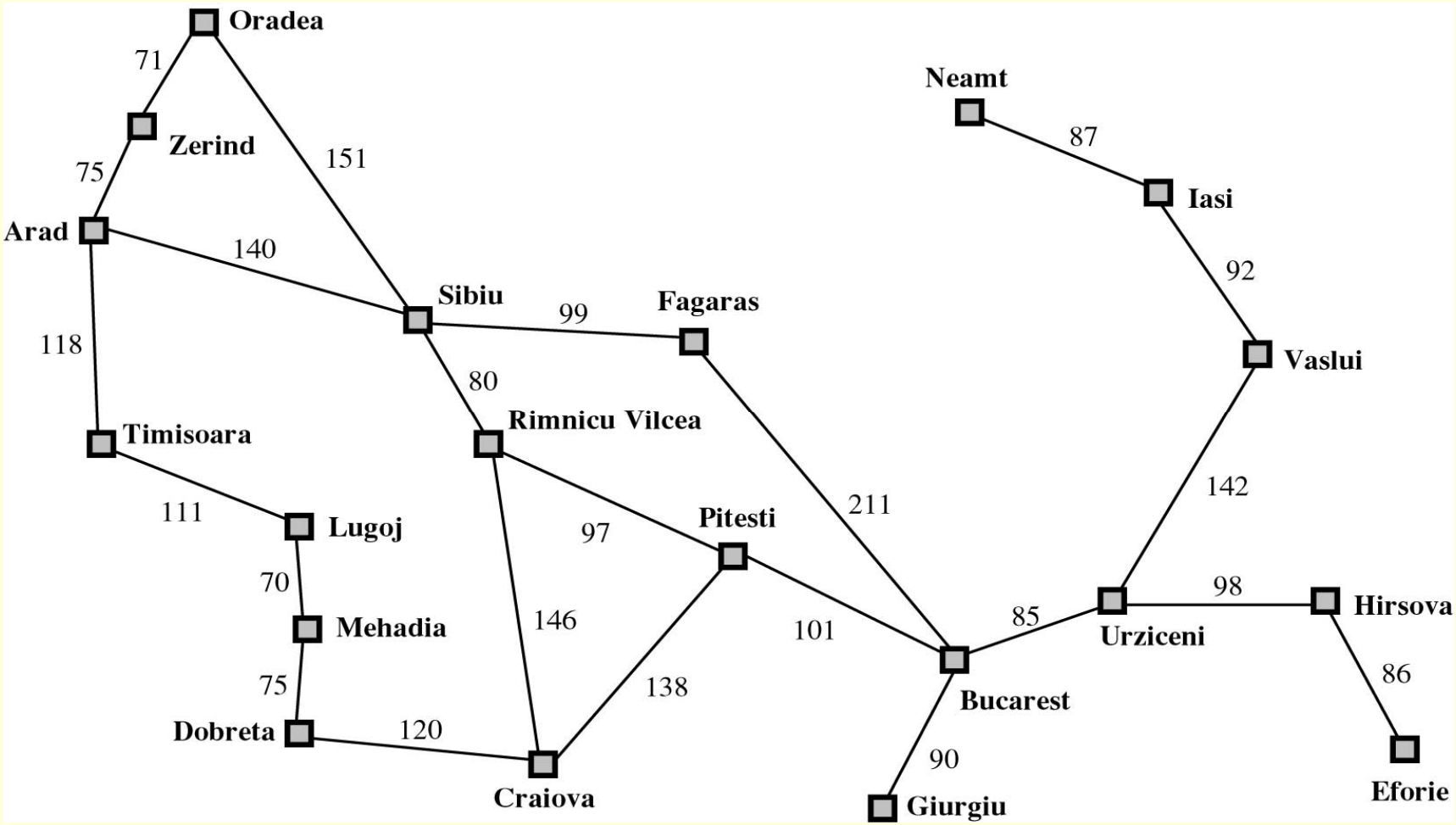
Después debemos decidir que estados y acciones considerar, esto es la:

Formulación del problema: {*estados*: varias ciudades, *acciones*: conducir entre las ciudades}

Finalmente necesitamos encontrar una secuencia de acciones que conduzcan a la solución, esto se llama *búsqueda:*

Encontrar solución: Arad, Sibiu, Fagaras, Bucarest.

Agente de vacaciones



Sencillo agente resolvente de problemas:

función AGENTE-SENCILLO-RESOLVENTE-PROBLEMAS(percepción) **devuelve** una acción

entradas: *percepción*, una percepción

estático: *sec*, una secuencia de acciones, vacía inicialmente

estado, una descripción del estado actual del mundo

objetivo, un objetivo inicialmente nulo

problema, una formulación del problema

estado ← ACTUALIZAR-ESTADO(*estado, percepción*)

si *sec* está vacía **entonces hacer**

objetivo ← FORMULAR OBJETIVO(*estado*)

problema ← FORMULAR-PROBLEMA(*estado, objetivo*)

sec ← BÚSQUEDA(*problema*)

acción ← PRIMERO(*secuencia*)

sec ← RESTO(*secuencia*)

devolver *acción*

Se está suponiendo que el agente es :

Estático **Observable** **Discreto** **Determinista**

Problemas y soluciones bien definidos

Un problema se define formalmente mediante cuatro componentes:

- El *estado inicial* en que se encuentra el agente.

- Una *descripción de las posibles acciones* disponibles por el agente, comúnmente se hace mediante la *función sucesor*, el espacio de estados viene implícito por el estado inicial y la función sucesor, el espacio de estados forma un grafo donde los nodos son los estados y los arcos son las acciones

- *El test objetivo* el cual determina si un estado es un estado objetivo

- Una función llamada costo del camino que a cada camino le asigna un valor, en costo numérico. Se supondrá que el costo del camino se obtiene mediante la suma de los costos individuales

$$\text{Costo del camino} = \sum_{i=1}^{j} c(x_i, a_i, y_i) \quad x_i, y_i \in \{Estados\}\ a_i \in \{acciones\}$$

Problemas de juguete

El mundo de la aspiradora

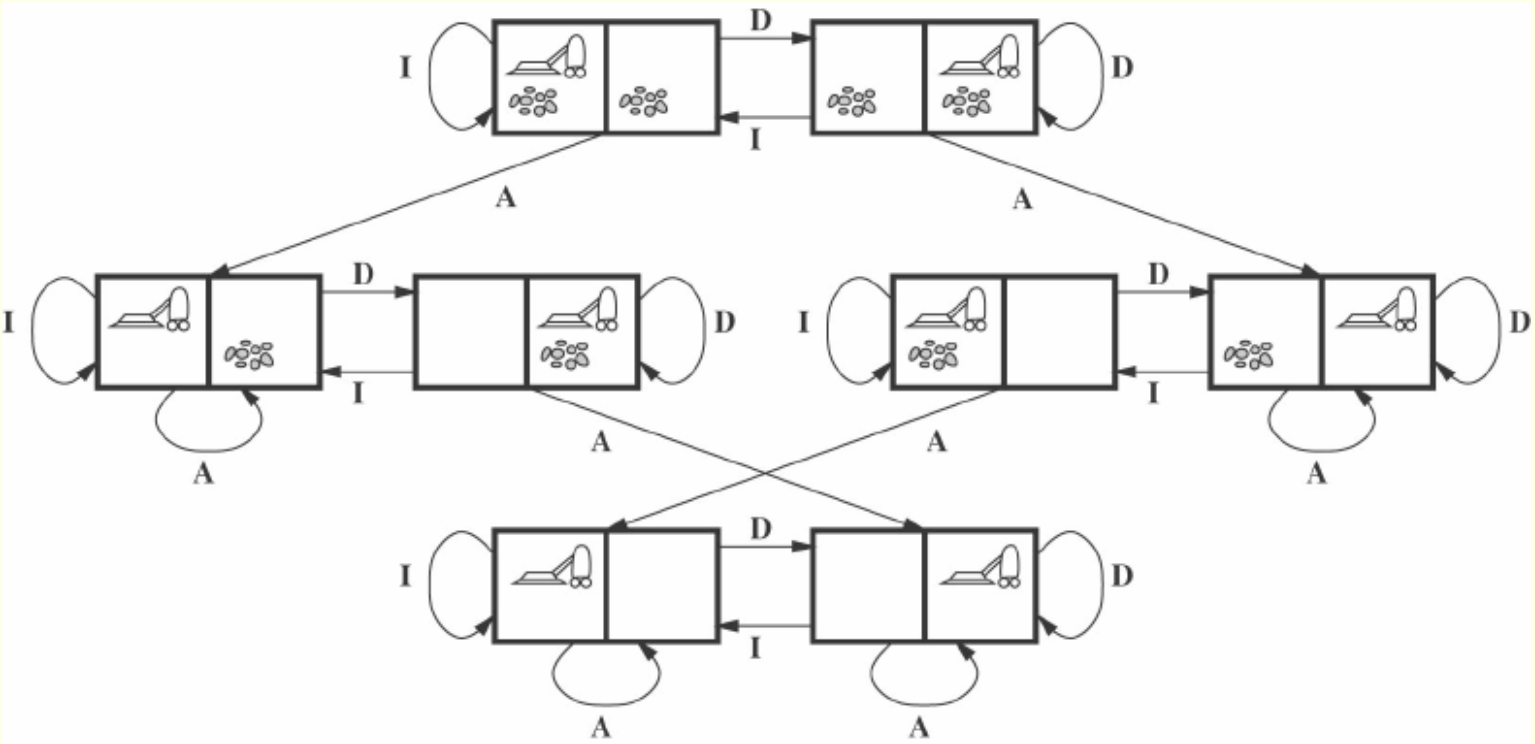
Estados: ***(posición, estado de A, estado de B)***

Estado inicial: **Cualquiera**

Función sucesor: **Genera los estados resultantes al aplicar Izq, Der, Asp**

Test objetivo: **Comprobar si todos los cuadrados están limpios**

Costo del camino: **Cada costo individual es 1, así el costo total es el número de pasos**



Problemas de juguete

El 8 Puzzle

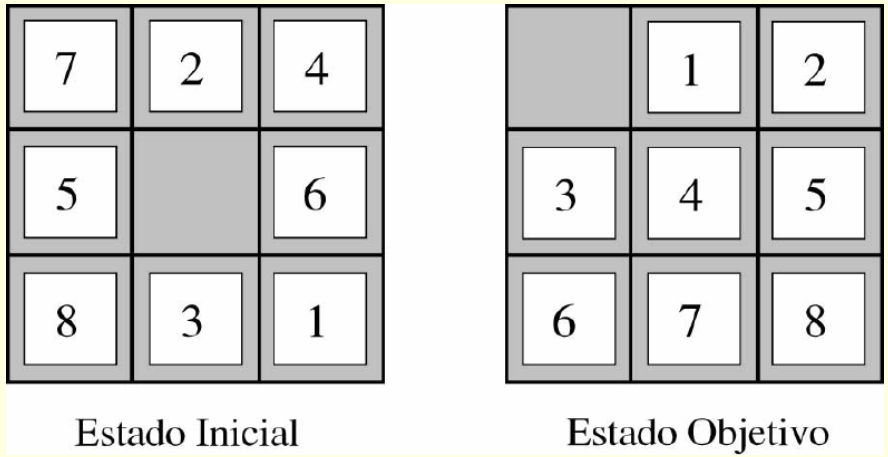
Estados: La descripción de un estado especifica la localización de cada una de las ocho fichas y la posición del espacio en blanco

Estado inicial: Cualquiera

Función sucesor: Genera los estados resultantes al aplicar Izquierda, Derecha, Arriba, Abajo

Test objetivo: Comprobar si el estado coincide con la configuración objetivo

Costo del camino: Cada costo individual es 1, así el costo total es el número de pasos



Estado Inicial Estado Objetivo

Pertenece a la clase de programas NP completos

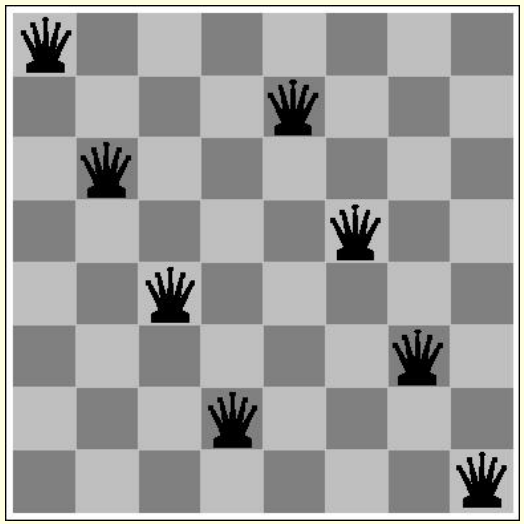
El 8-puzzle tiene 9!/2=181,440 estados alcanzables
El 15-puzzle tiene 1.3 trillones de estados y se resuelve en milisegundos
El 24-puzzle tiene alrededor de 10^{25} estados y no es posible resolverlo de manera optima

Problemas de juguete

Las 8 reinas (formulación incremental)

Se comienza con el tablero vació y se van añadiendo reinas una a una

Estados:	**Cualquier combinación de 0 a 8 reinas en un tablero de ajedrez**
Estado inicial:	**Cero reinas colocadas**
Función sucesor:	**Añade una reina en cualquier cuadro vacío**
Test objetivo:	**8 reinas en el tablero y ninguna es atacada**
Costo del camino:	**Cada adición de una reina tiene costo unitario, el costo siempre es 8.**



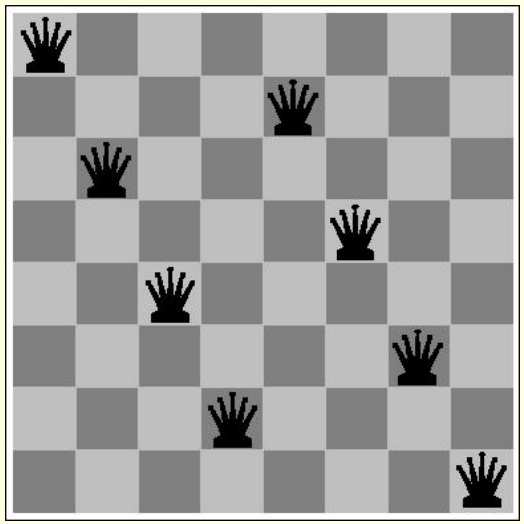
El espacio de estados es de tamaño $64x63x\ldots x57 \approx 3X10^{14}$

Problemas de juguete

Las 8 reinas (formulación completa)

Se comienza con las 8 reinas en el tablero

Estados:	**Cualquier combinación de 8 reinas en un tablero de ajedrez**
Estado inicial:	**La combinación de 8 reinas una por columna**
Función sucesor:	**Añadir una reina en cualquier cuadrado en la columna más a la izquierda vacía, tal que no sea atacada por cualquier otra reina.**
Test objetivo:	**8 reinas en el tablero y ninguna es atacada**
Costo del camino:	**Cada adición de una reina tiene costo unitario, el costo siempre es 8.**



El espacio de estados se reduce a 2 057

Problemas del mundo real

Lineas aéreas

Estados: Cada estado está representado por una localización y la hora.

Estado inicial: Especificado por el problema.

Función sucesor: Devuelve los estados que resultan de tomar cualquier vuelo desde un aeropuerto a otro, considerando la hora y el tiempo de viaje.

Test objetivo: ¿Tenemos nuestro destino para una cierta hora específica?

Costo del camino: Costo en dinero, tiempo de espera, tiempo de vuelo, calidad del asiento, tipo de avión.

Problema turístico (visitar cada ciudad al menos una vez)

Estados: Cada estado debe incluir localización y ciudades visitadas.

Estado inicial: Cualquier cuidad.

Función sucesor: Viajar a una ciudad adyacente

Test objetivo: ¿He visitado todas las ciudades?

Costo del camino: Costo en dinero, tiempo.

Problema agente viajero (visitar cada ciudad exactamente una vez)

Búsqueda de soluciones

La solución de problemas se hace mediante la búsqueda en el espacio de estados, para lo cual necesitamos de técnicas de búsqueda. Las técnicas que se presentan aquí utilizan un árbol de búsqueda explicito generado por el estado inicial y la función sucesor.

La raíz del árbol es el nodo de búsqueda que corresponde al estado incicial.

Expandir es aplicar la función sucesor a algún estado y generar un nuevo conjunto de estados

De manera general un algoritmo de búsqueda sigue tres pasos:

Escoger Comprobar Expandir

El estado a expandir se selecciona mediante la estrategia de búsqueda.

El espacio de estados se diferencia del árbol de búsqueda en que este último tienen las soluciones y puede ser infinito.

Descripción informal del ALGBA

función BÚSQUEDA-ÁRBOLES(*problema*,*estrategia*) **devuelve** una solución o fallo

inicializa el árbol de búsqueda usando el estado inicial del *problema*

Hacer ciclo

si no hay candidatos para expandir **entonces devolver** fallo
escoger, de acuerdo a la *estrategia*, un nodo hoja para expandir
si el nodo contiene un estado objetivo **entonces devolver** la correspondiente solución
en otro caso expandir el nodo y añadir los nodos resultado al árbol de búsqueda

Estados vs nodos

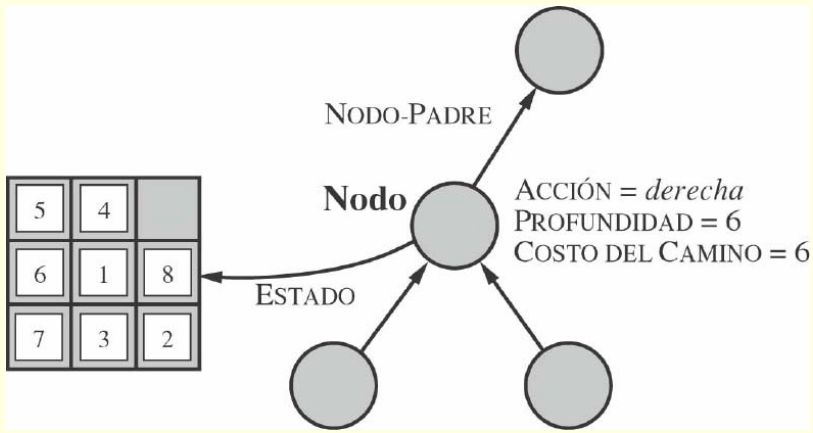
Un *estado* es una (representación de) una configuración física.

Un *nodo* es una estructura de datos que forma parte de un árbol de búsqueda que incluye:

- **Estado** — **El estado que corresponde al nodo.**
- **Nodo padre** — **El nodo en el árbol de búsqueda que ha generado este nodo**
- **Acción** — **La acción que se aplico al nodo padre para generar este**
- **Costo del camino** — **Denotado por g(n) es el costo desde la raíz hasta el nodo**
- **Profundidad** — **Número de pasos a lo largo del camino**

¡Los *estados* no tienen padres, hijos, profundidad o costo del camino!

Frontera: es la colección de nodos no comprobados llamados nodos hoja.



Descripción menos informal del ALGBA

Suponiendo que la colección de nodos es una cola.

HACER-COLA(*elemento*$_1$,*elemento*$_2$,…*elemento*$_n$)	crea una cola con los elementos
VACIA?(*cola*)	devuelve verdadero o falso
PRIMERO(*cola*)	devuelve el primer elemento de la cola
BORRAR-PRIMERO(*cola*)	devuelve el primer elemento de la cola y lo borra de la cola
INSERTA(*elemento,cola)*	inserta un elemento a la cola
INSERTA-TODO(*elementos,cola)*	inserta una colección de elementos a la cola

Podemos ahora escribir una versión más formal del ALGBA

función BÚSQUEDA-ÁRBOLES(*problema*,*frontera*) **devuelve** una solución o fallo

frontera ←INSERTA(HACER-NODO(ESTADO-INICIAL[*problema*], *frontera*))

inicializa el árbol de búsqueda usando el estado inicial del *problema*

Hacer bucle

si VACIA?(*frontera*) **entonces devolver** fallo

nodo ← BORRAR-PRIMERO*(frontera)*

si TEST-OBJETIVO*[problema]* aplicado al estado *nodo* es cierto

entonces devolver SOLUCION(*nodo*)

frontera ←INSERTAR-TODO(EXPANDIR*(nodo,problema), frontera*)

Medir el rendimiento de una estrategia de búsqueda

Si a un algoritmo se le pide que encuentre la solución de un problema, puede dar como resultado fallo o solución en el mejor de los casos ya que también puede suceder que al intentar encontrar la solución entre en un bucle infinito, entonces para medir el rendimiento de un algoritmo se consideran 4 factores:

- **Completitud** — ¿Si existe una solución está garantizado que el algoritmo la encuentre?
- **Optimización** — ¿Encuentra el algoritmo la solución optima, de acuerdo a la función costo del camino?
- **Complejidad en tiempo** — ¿Cuánto tarda en encontrar una solución?
- **Complejidad en espacio** — ¿Cuánto memoria se necesita para el funcionamiento de la búsqueda?

La complejidad en tiempo y espacio se considera de acuerdo a alguna medida de la dificultad del problema, en IA se expresa en términos de:

1. El factor de ramificación, que es el máximo número de sucesores de cualquier nodo **b**
2. La profundidad del nodo objetivo más superficial **d**
3. La longitud máxima de cualquier camino **m**

El tiempo a menudo se mide como el número de nodos generados durante la búsqueda, y el espacio se mide en términos del máximo número de nodos que se almacenan en la memoria.

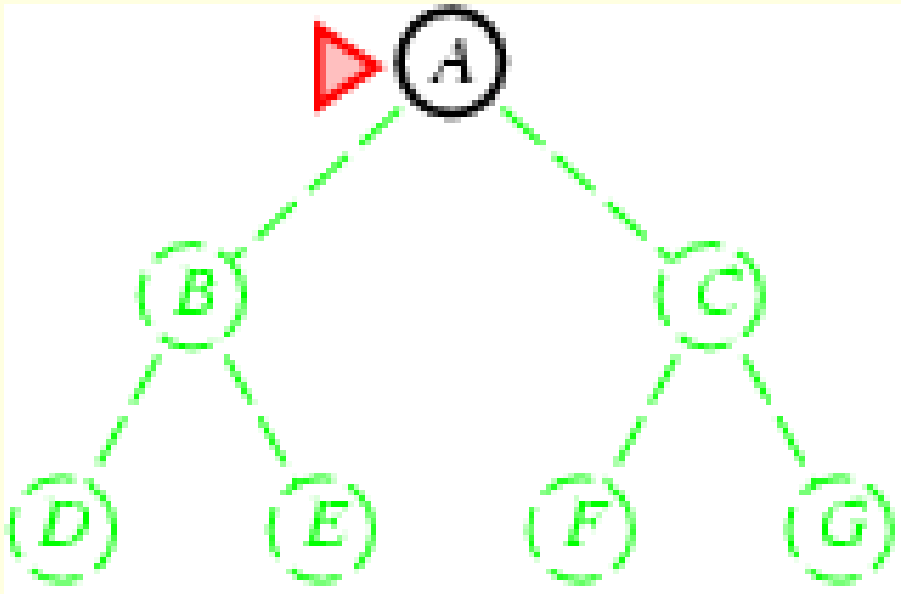
Estrategias de búsqueda no informada

Las estrategias de búsqueda no informada sólo utilizan la información disponible en la definición del problema, todo lo que se puede hacer es generar los sucesores y comprobar si se trata de un esatdo objetivo o no.

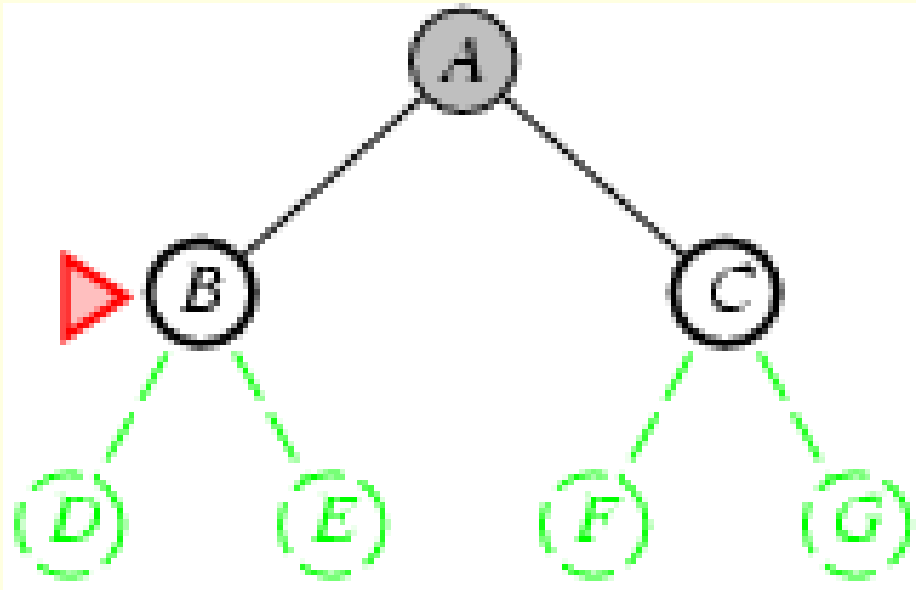
- Búsqueda primero en anchura. BPA
- Búsqueda con costo uniforme. BCU
- Búsqueda primero en profundidad. BPP
- Búsqueda de profundidad limitada. BPL
- Búsqueda con profundidad iterativa. BPI
- Búsqueda bidireccional BB

Búsqueda primero en anchura

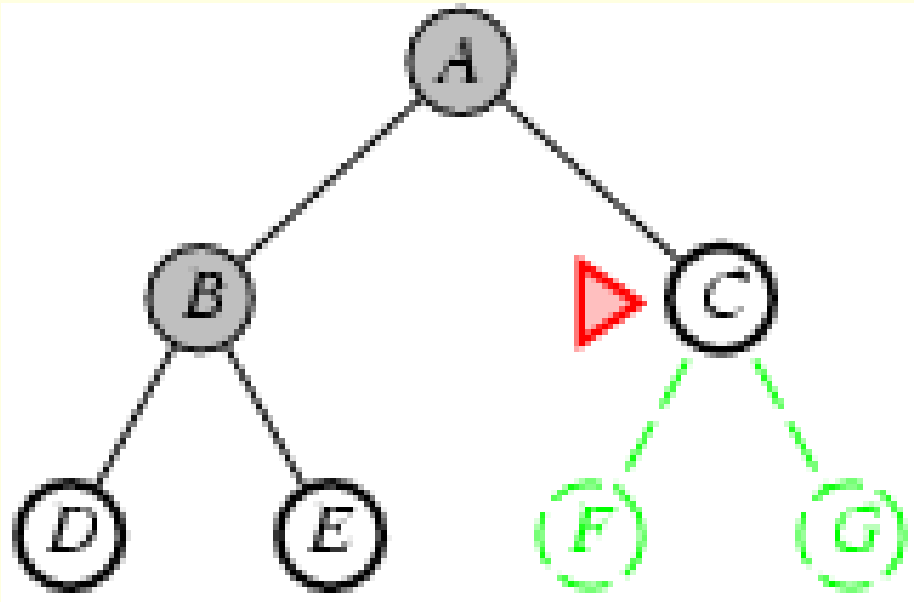
- Se expande primero el nodo raíz, a continuación, se expanden todos los sucesores del nodo raíz, después sus sucesores, etc.
- Implementación:Se usa una estructura FIFO, es decir, el primero en entrar es el primero en salir.



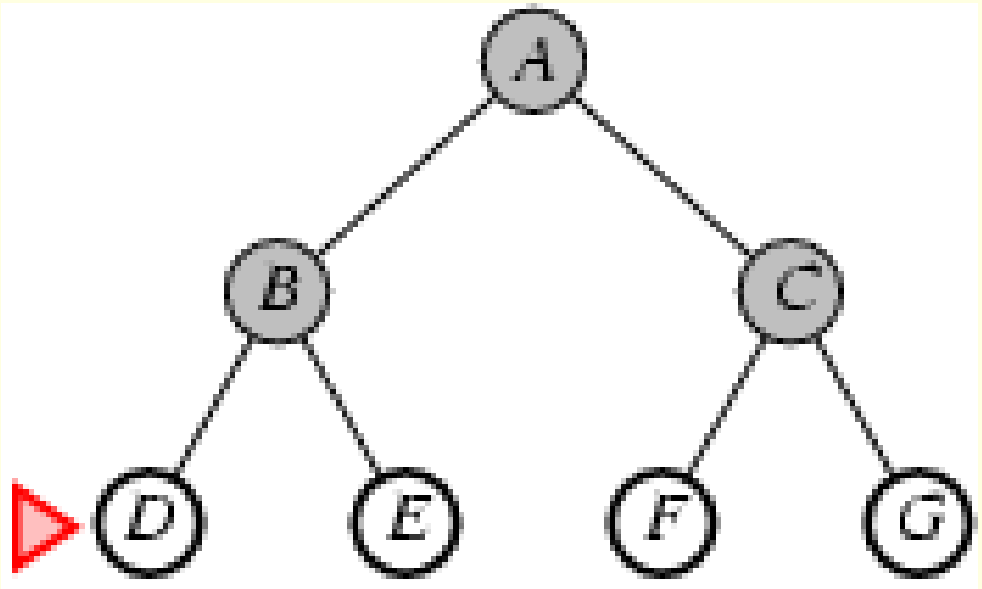
Búsqueda primero en anchura



Búsqueda primero en anchura



Búsqueda primero en anchura



Propiedades de la búsqueda primero en anchura

- ¿Completa? Sí (si *b* es finito)

Supongamos un factor de ramificación b y si la solución está a una profundidad d, entonces el máximo número de nodos generados es:

$$1+b+b^2+b^3+\ldots+b^d + b(b^d-1)$$

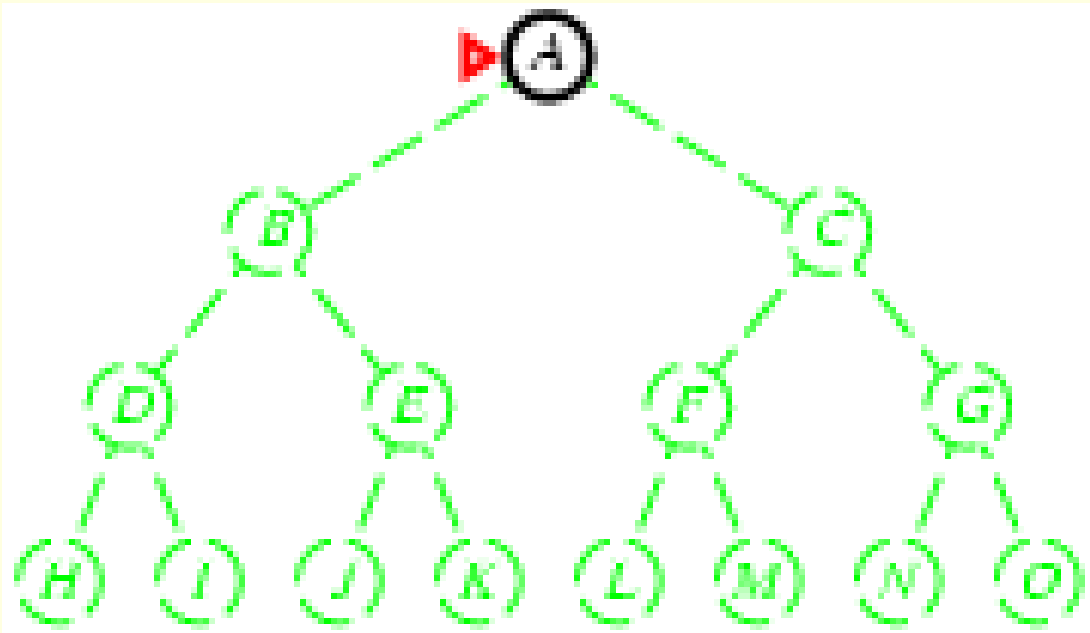
Por lo que se tiene que

- ¿Tiempo? Es del orden $O(b^{d+1})$
- ¿Espacio? Igual que el tiempo tiene orden $O(b^{d+1})$ pues mantiene todos los nodos en la memoria)
- ¿Óptima? Será optima si el costo del camino es una función de d no decreciente

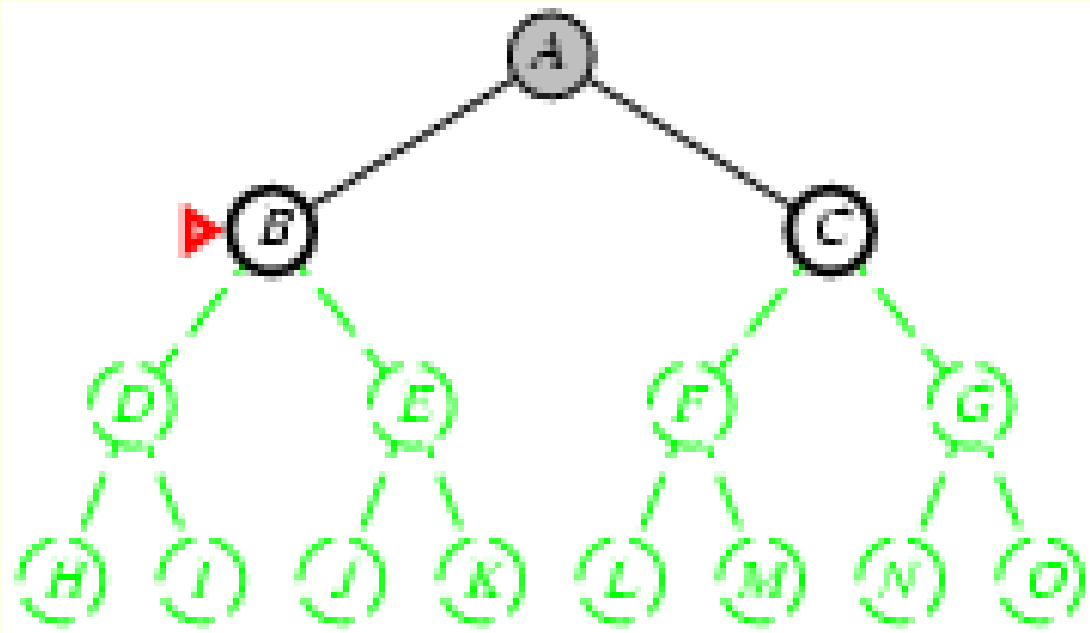
Su principal defecto es la cantidad de memoria necesaria, por ejemplo si b=d=10 entonces requeriría de 101 TERABYTES!

Búsqueda primero en profundidad

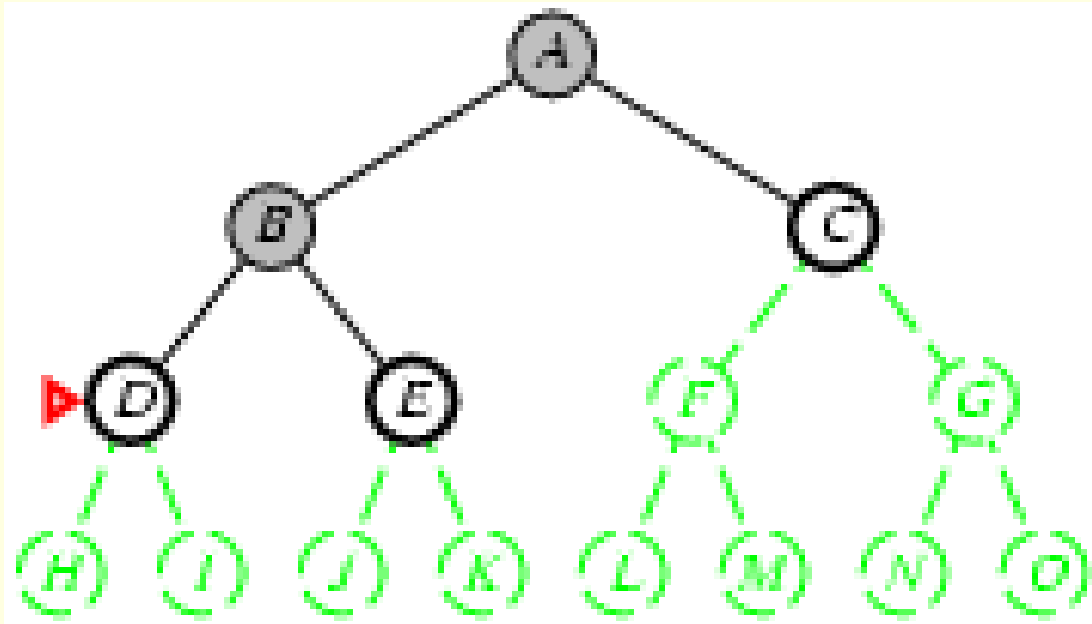
- Se expande el nodo no expandido más profundo.
- Implementación:Usa una estructura LIFO, los últimos en entrar son los primeros es salir



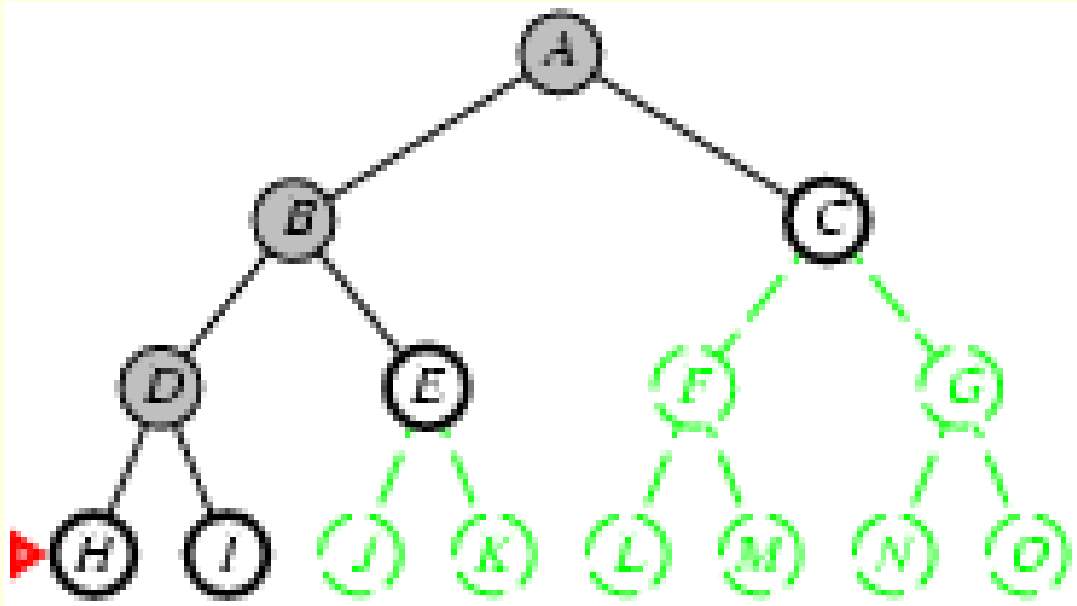
Búsqueda primero en profundidad



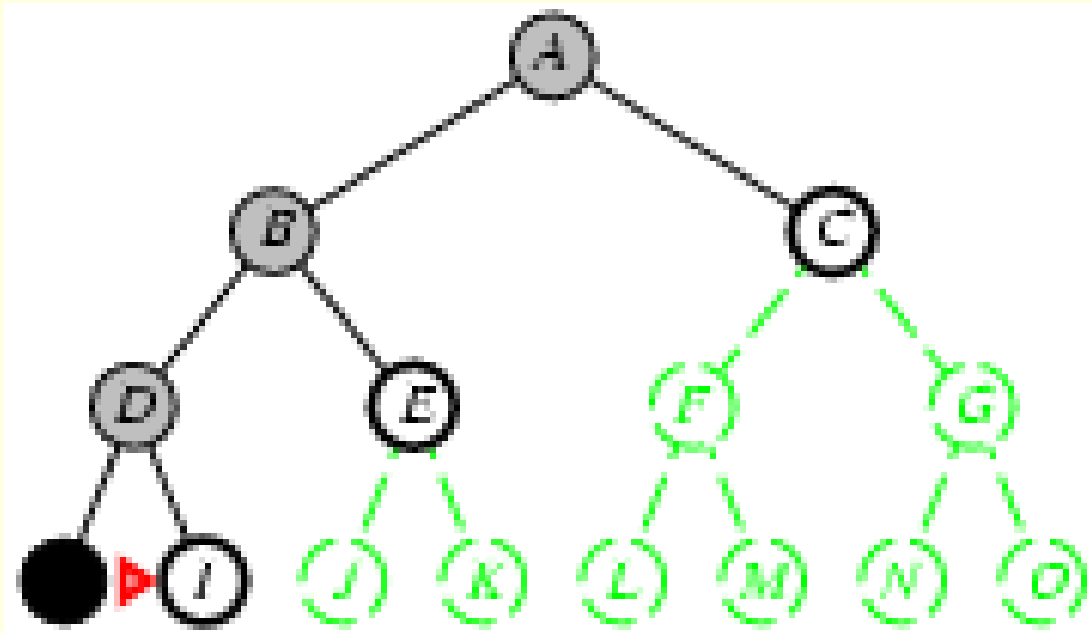
Búsqueda primero en profundidad



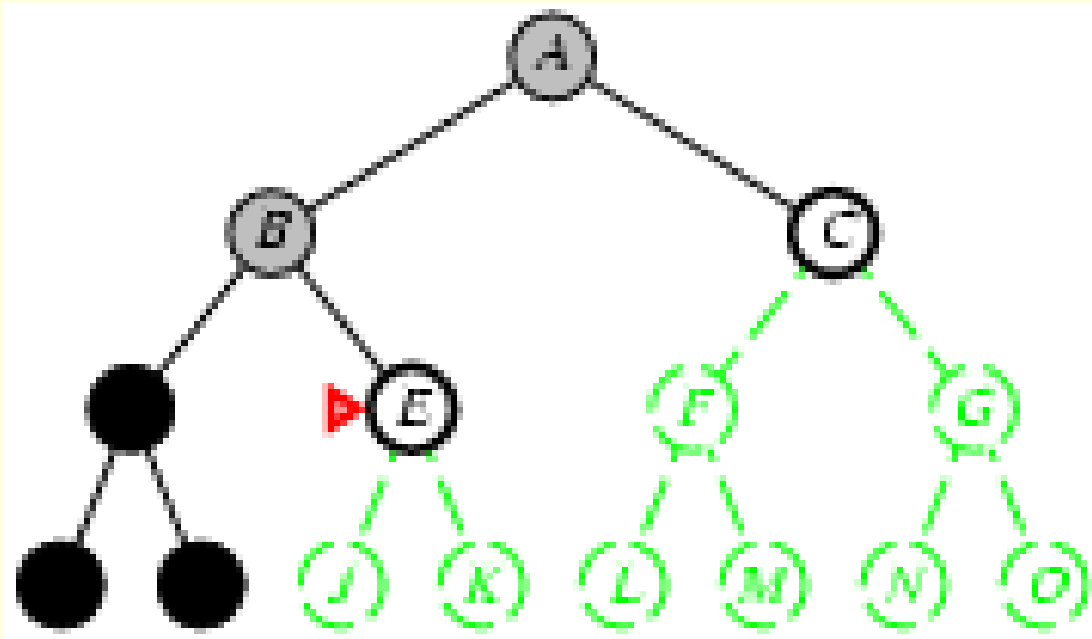
Búsqueda primero en profundidad



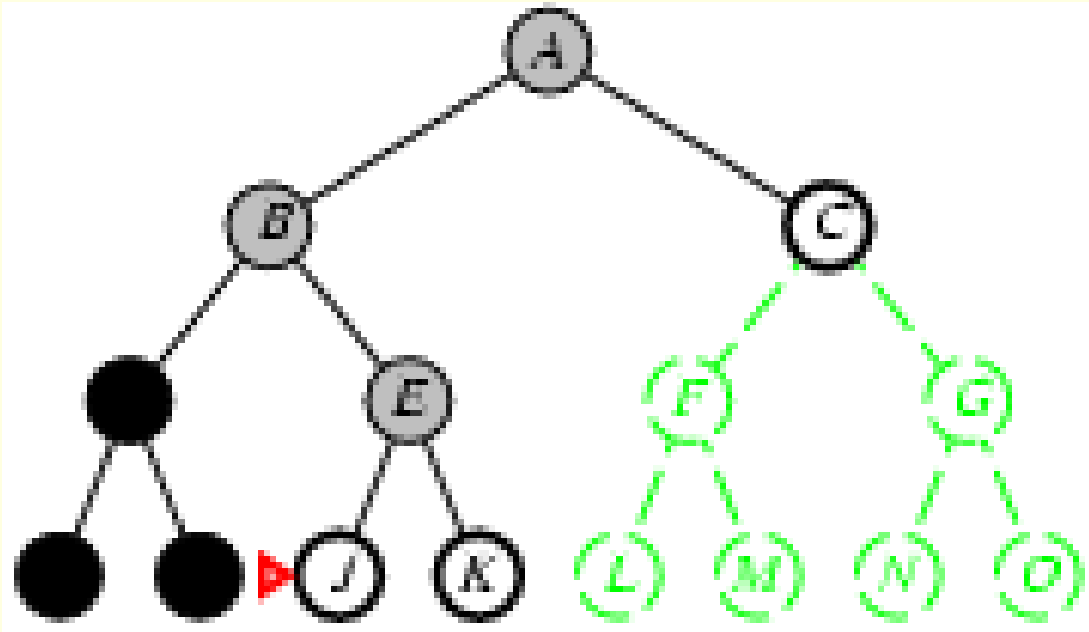
Búsqueda primero en profundidad



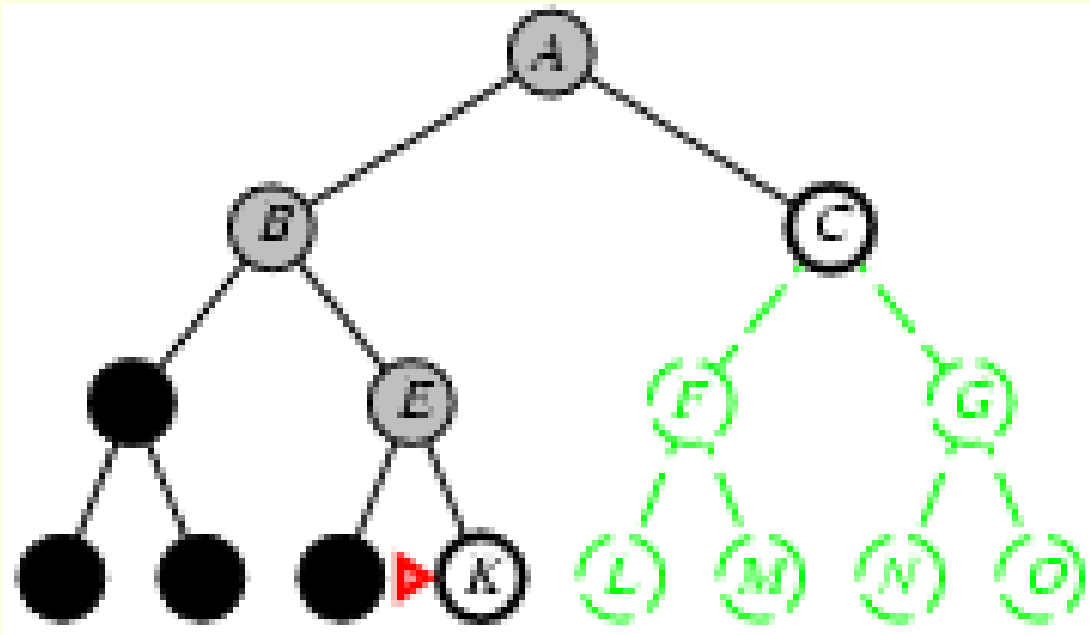
Búsqueda primero en profundidad



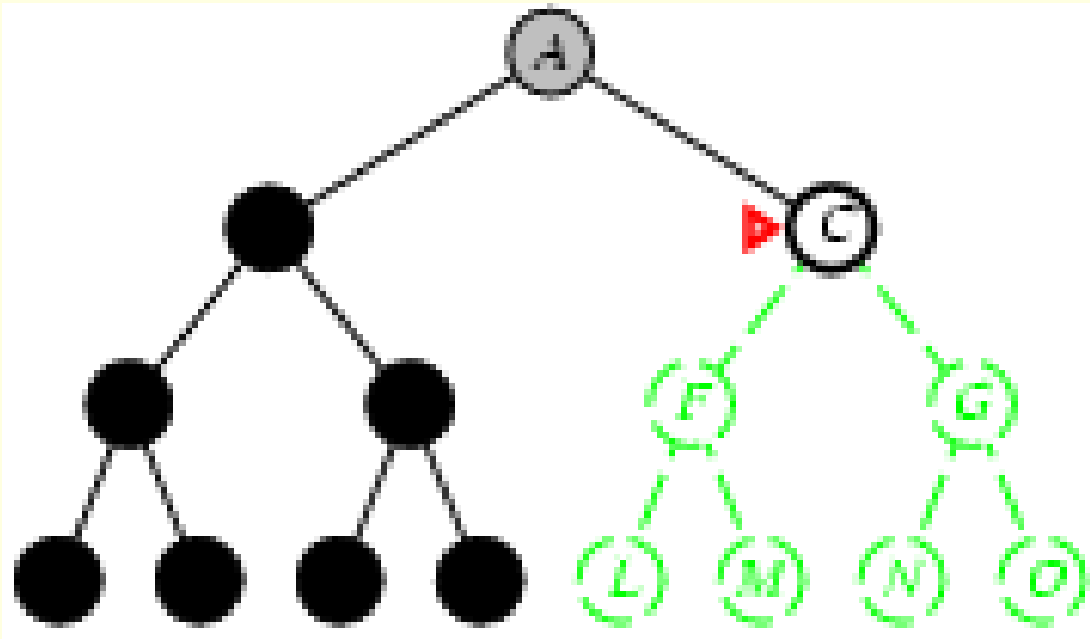
Búsqueda primero en profundidad



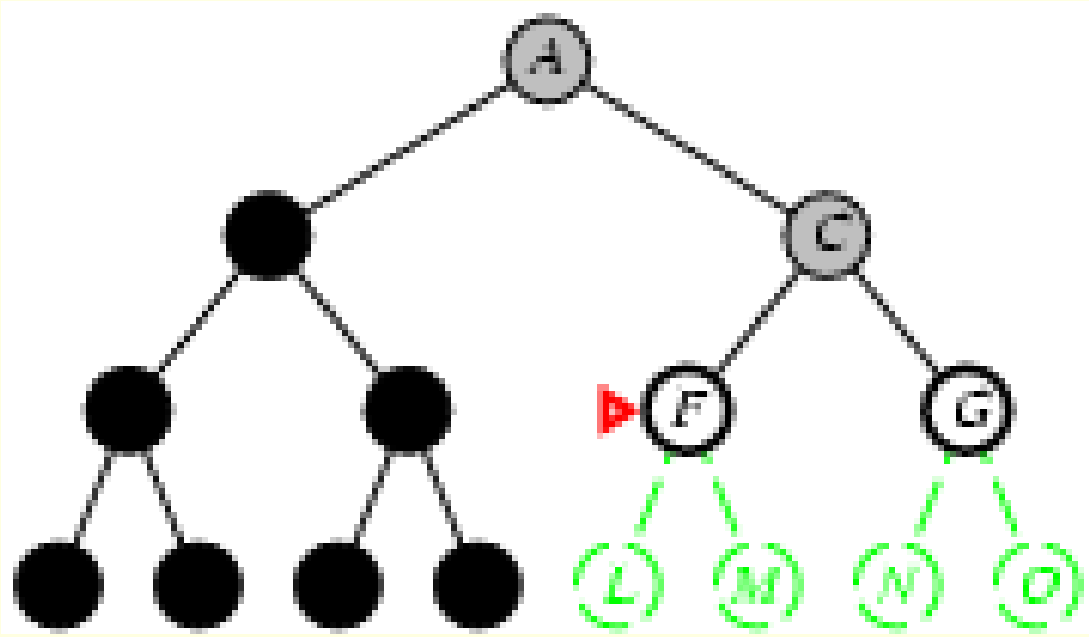
Búsqueda primero en profundidad



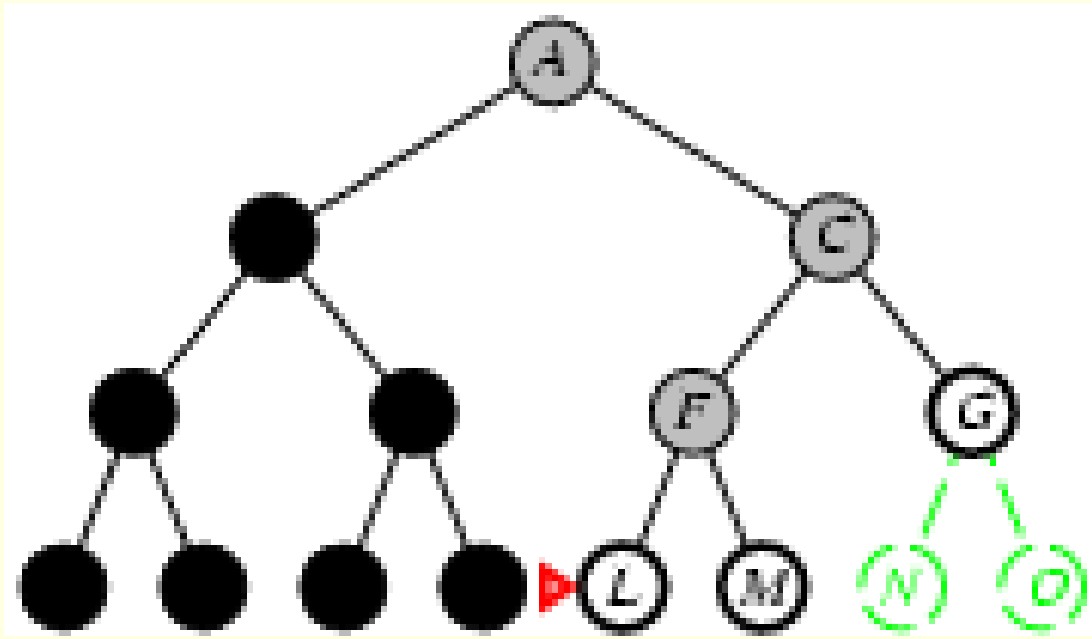
Búsqueda primero en profundidad



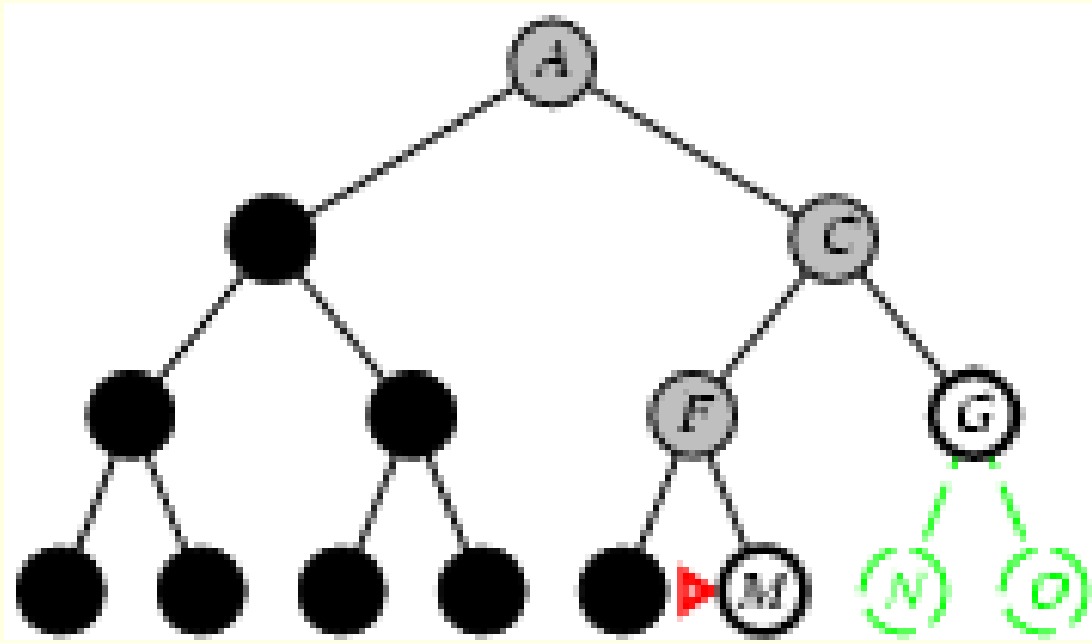
Búsqueda primero en profundidad



Búsqueda primero en profundidad



Búsqueda primero en profundidad



Propiedades de la búsqueda primero en profundidad

Este tipo de búsqueda consume mucha menos memoria que la BPA, ya que sólo almacena un camino desde la raíz a un nodo hoja, junto con los nodos hermanos no expandidos.

Para un espacio de estados con factor de ramificación b y profundidad máxima m, se necesita almacenar sólo bm+1 nodos, mientras que la BPA con b=10 y d=12 necesita 10 petabytes(10 000 000 gigabytes) de memoria la BPP necesita 118 kilobytes, es decir **10 billones veces menos espacio**

El inconveniente de la BPP es que puede hacer una elección equivocada y obtener un camino muy largo inclusive infinito por ejemplo si el nodo objetivo fuera C, primero se explorará el subárbol izquierdo y luego el derecho que es en donde esta C, si J fuera también un nodo objetivo la BPP diría que esa es la solución, por lo tanto no es óptima y si el subárbol fuera de profundidad ilimitada y no contuviera solución no terminaría por lo tanto tampoco es completa.

Búsqueda de profundidad limitada BPL

Para solventar los problemas de la BPP podemos fijar un número que funcione como un límite de profundidad digamos L, pensaremos que los nodos a profundidad L ya no tienen sucesores, con esto se resuelve el problema del camino infinito, pero si:

L>d se vuelve incompleto y si

L<d no será optimo

Su complejidad en tiempo es $O(b^L)$ y

Su complejidad en espacio es O(b x L)

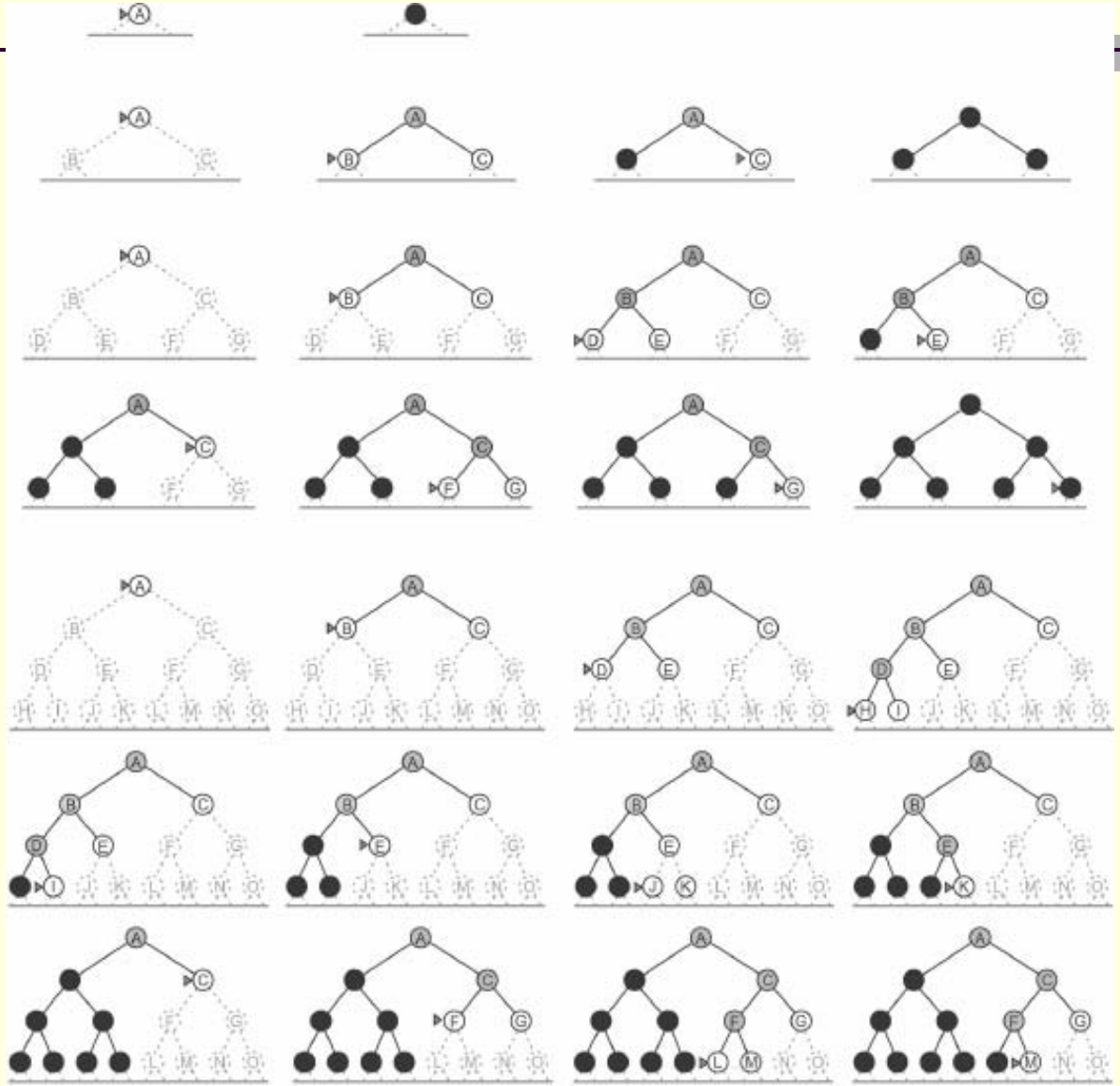
En algunos problemas se puede conocer algo sobre L pero en la mayoría no, por ejemplo en el caso del agente viajero L=19 sería algo razonable ya que sólo hay 20 ciudades, sin embargo analizando el mapa veríamos que L=9 es suficiente

Búsqueda con profundidad iterativa BPI

La BPI es una estrategia usada en combinación con la BPP que encuentra el mejor límite de profundidad aumentando gradualmente el límite 0,1,2,… hasta encontrar un objetivo. La profundidad iterativa combina las ventajas de la BPP y la BPA

La BPI es el método de búsqueda no informada preferido cuando hay un espacio grande de búsqueda y no se conoce la profundidad de la solución

Cuatro iteraciones de la BPI sobre un árbol binario



Cuadro comparativo de búsquedas no informadas

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bi-directional
Complete	Yes[1]	Yes[1,2]	No	No	Yes[1]	Yes[1,4]
Time	$O(b^{d+1})$	$O\left(b^{\left\lceil \frac{C^*}{\varepsilon} \right\rceil}\right)$	$O(b^m)$	$O(b^l)$	$O(b^d)$	$O\left(b^{\frac{d}{2}}\right)$
Space	$O(b^{d+1})$	$O\left(b^{\left\lceil \frac{C^*}{\varepsilon} \right\rceil}\right)$	$O(bm)$	$O(bl)$	$O(bd)$	$O\left(b^{\frac{d}{2}}\right)$
Optimal	Yes[3]	Yes	No	No	Yes[3]	Yes[3,4]

Table 2.17: Complexities of uninformed search procedures.

[1] *if b is finite;*
[2] *if step costs $\geq \varepsilon$ for positive ε;*
[3] *if step costs are all identical;*
[4] *if both directions use breadth-first search*

Symbols:

- b *branching factor*
- d *depth of the shallowest solution*
- m *maximum depth of the search tree*
- l *depth limit*
- C^* *cost of the optimal solution*

Evitar estados repetidos

¿Cómo evitar perder el tiempo revisando estados que ya se habían revisado?

- Para algunos problemas esta posibilidad nunca aparece
- Para algunos otros es inevitable (acciones reversibles)

Si un algoritmo no detecta los estados repetidos estos pueden llegar a provocar que un problema soluble llegue a ser insoluble.

Para la BPP, los únicos nodos en memoria so aquellos del camino desde la raíz hasta el nodo actual. La comparación de estos nodos permite al algoritmo descubrir los caminos que forman ciclos y que pueden eliminarse inmediatamente

Se puede modificar el ALGBA incluyendo una estructura de datos llamada lista cerrada, que almacene cada nodo expandido. SI el nodo actual de la lista se empareja con un nodo de la lista cerrada en vez de expandirlo se elimina, el nuevo algoritmo se le llama algoritmo general de la búsqueda en grafos ALGBG

En problemas con muchos estados repetidos ALGBG es superior a ALGBA