
Inteligencia Artificial

Solución de problemas mediante Búsquedas

Jorge Luis Guevara Diaz

www.jorge.sistemasyservidores.com

Que veremos hoy?

1. Agentes solucionadores de problemas
2. Formulación de problemas
3. Ejemplo de problemas
4. Algoritmos Básicos de Búsquedas

Agentes solucionadores de problemas

Agentes solucionadores de problemas

- Cuatro pasos generales para solucionar problemas
 - **Formulacion de la Meta (Goal)**
 - Cuales son los estados del mundo satisfactorios
 - **Formulacion de problema (problem)**
 - Que acciones y estados se deben considerar dada la meta
 - **Búsqueda (search)**
 - Determinar la posible secuencia de acciones que conducen a estados de valores conocidos y luego escoger la mejor secuencia
 - **Ejecución (Execute)**
 - Dada la solución realizar las acciones

Agentes solucionadores de problemas

Ejemplo: Rumania

De vacaciones en Rumania; Actualmente en Arad

- El vuelo sale mañana a Bucarest

Formular Meta

- Estar en Bucarest

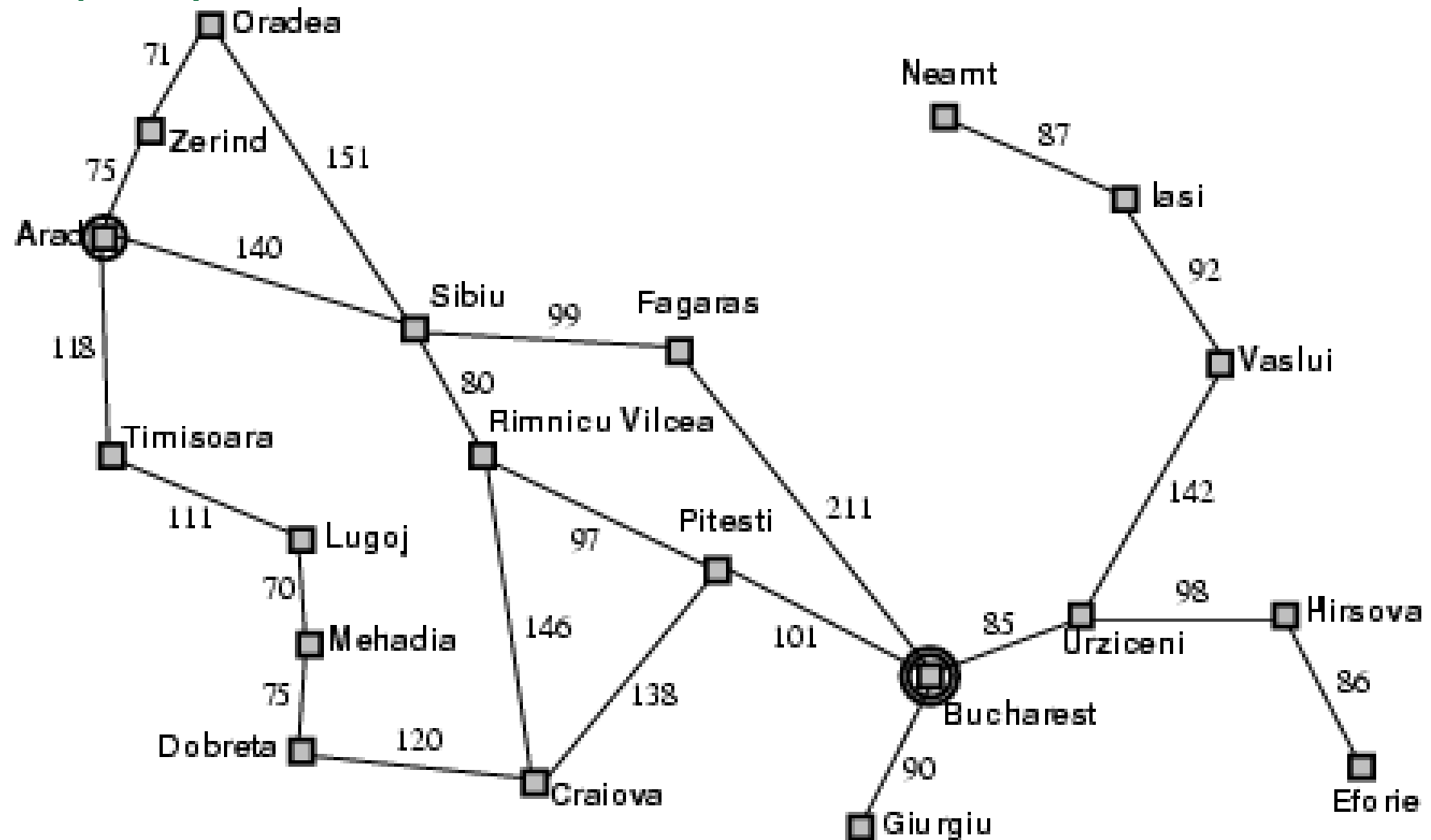
Formular problema

- Estados : varias ciudades
- Acciones : manejar entre ciudades

Encontrar solución

- Secuencia de Ciudades; Ejem. Arad, Sibiu, Fagaras, Bucharest, ...

Agentes solucionadores de problemas ejemplo: Romania



Agentes solucionadores de problemas

Algoritmo

```
function SIMPLE-PROBLEM-SOLVING-AGENT(percept) returns an action
  static: 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 do
    goal ← FORMULATE-GOAL(state)
    problem ← FORMULATE-PROBLEM(state, goal)
    seq ← SEARCH(problem)
  action ← FIRST(seq)
  seq ← REST(seq)
  return action
```

Formulación de problemas

Formulación de problemas

Un **problema** está definido por cuatro items:

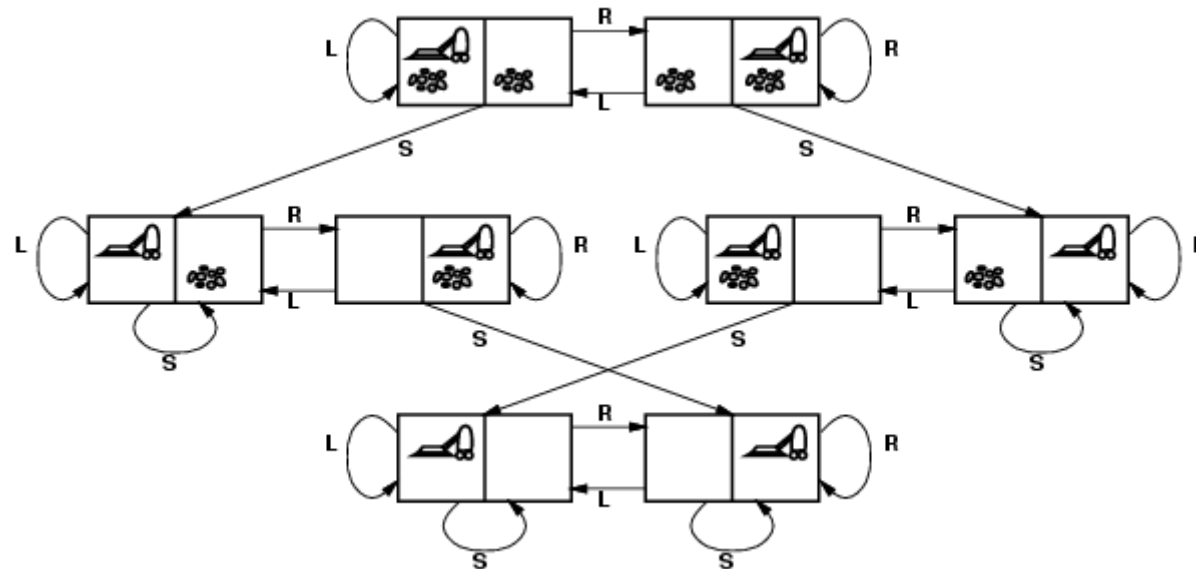
1. **Estado inicial** ejem., “en Arad” □
 2. **función sucesor** $S(x)$ = conjunto de pares acción–estado
 - ejem, $S(\text{Arad}) = \{ \langle \text{Arad} \rightarrow \text{Zerind}, \text{Zerind} \rangle, \dots \}$.
 - **Espacio de estados** : conjunto de todos los estados alcanzables desde el estado inicial
 - **Camino** : secuencia de estados conectados por secuencia de acciones
 3. **Test meta**, es el (los) estado x el (los) estado meta? puede ser :
 - **explícito**, ejem, $x = \text{“en Bucharest”}$
 - **implícito**, ejem, $\text{jaquemate}(x)$
 4. **costo del camino** (aditivo)
 - ejem, suma de distancias, numero de acciones ejecutadas, etc.
 - $c(\text{estado } x, \text{accion } a, \text{estado } y)$ es el **costo de paso**, se asume ser ≥ 0
- Una **solución** es una secuencia de acciones que van del estado inicial hacia el estado meta

Seleccionando un espacio de estados

- El mundo real es complejo
 - el espacio de estados debe ser **abstraído** para solucionar el problema
- (Abstracción) estados = conjunto de estados reales
- (Abstracción) acción = combinación compleja de acciones reales
 - ejem, "Arad → Zerind" representa un conjunto complejo de posibles rutas, desvíos, paradas, etc. Para garantizar la realización, **cualquier** estado real "en Arad" debe tener **algún** estado real "en Zerind"
- Para garantizar la realización, **cualquier** estado real "en Arad" debe tener **algún** estado real "en Zerind"
- (Abstracción) solución =
 - Conjunto de caminos reales que son soluciones en el mundo real
- Cada acción abstracta debería ser "mas sencilla" que el problema original

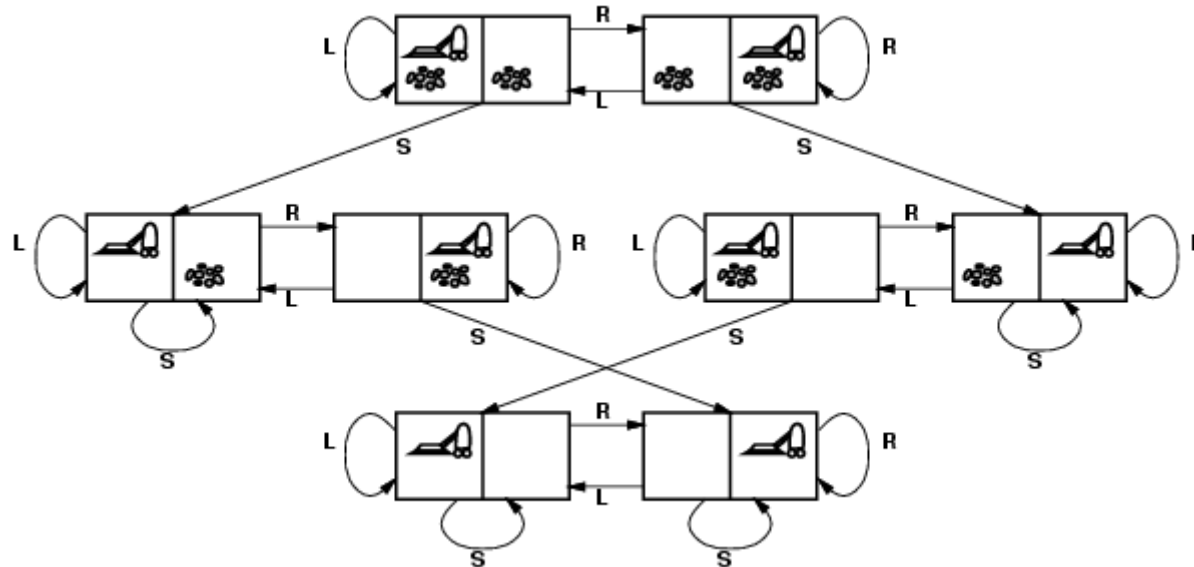
Ejemplo de problemas

aspiradora grafo de espacio de estados



- estados?
- estado inicial?
- acciones?
- test meta?
- costo del camino? □

aspiradora grafo de espacio de estados



- estados? Dos ubicaciones con o sin suciedad $2 \times 2^2=8$ estados.
- estado inicial? Cualquier estado puede ser inicial
- acciones? izquierda, Derecha, Aspirar
- test meta? Chekear que todos las ubicaciones esten limpios
- costo del camino? 1 por acción

ejemplo: El 8-puzzle

7	2	4
5		6
8	3	1

Start State

	1	2
3	4	5
6	7	8

Goal State

- estados?
- estado inicial?
- acciones?
- test meta?
- costo del camino?

ejemplo: El 8-puzzle

7	2	4
5		6
8	3	1

Start State

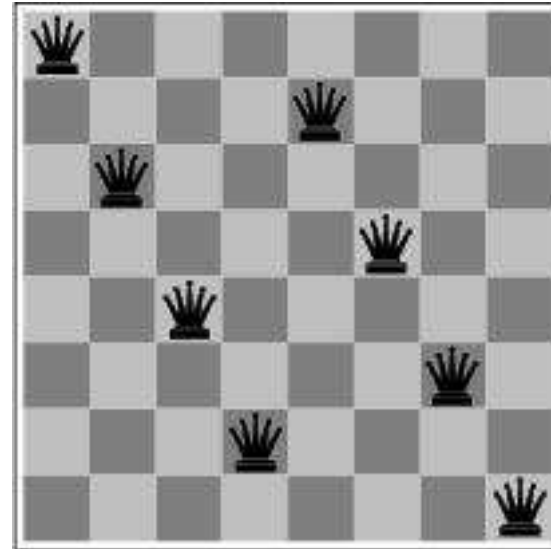
	1	2
3	4	5
6	7	8

Goal State

- estados? Ubicación de las piezas
- estado inicial? Cualquier estado puede ser inicial
- acciones? Mover cuadro en blanco arriba, abajo, derecha, izquierda.
- test meta? Estado meta (dado).
- costo del camino? 1 por movimiento.

Para las 8-reynas y para misioneros y canibales

- Estados
- estado inicial
- Acciones
- test meta
- costo del camino = $g(x)$



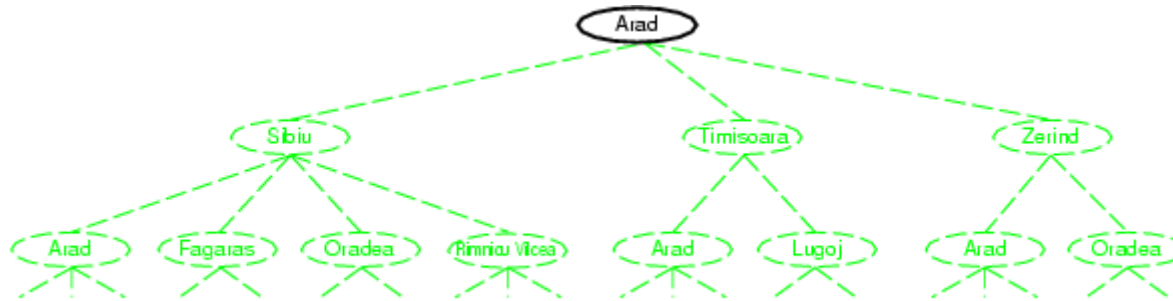
Algoritmos Básicos de Búsquedas

Algoritmos de Búsqueda en árboles

- **Idea básica:**

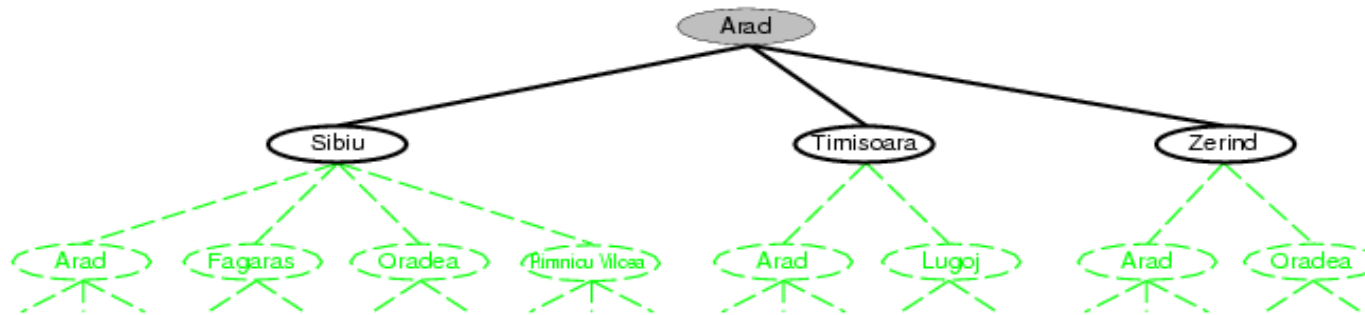
- Exploración del espacio de estados generando sucesores de los estados ya explorados
- Búsqueda a través de un árbol
Raíz = estado inicial
Nodos y hojas generados a través de la función sucesor

Ejemplo de árbol de Búsqueda



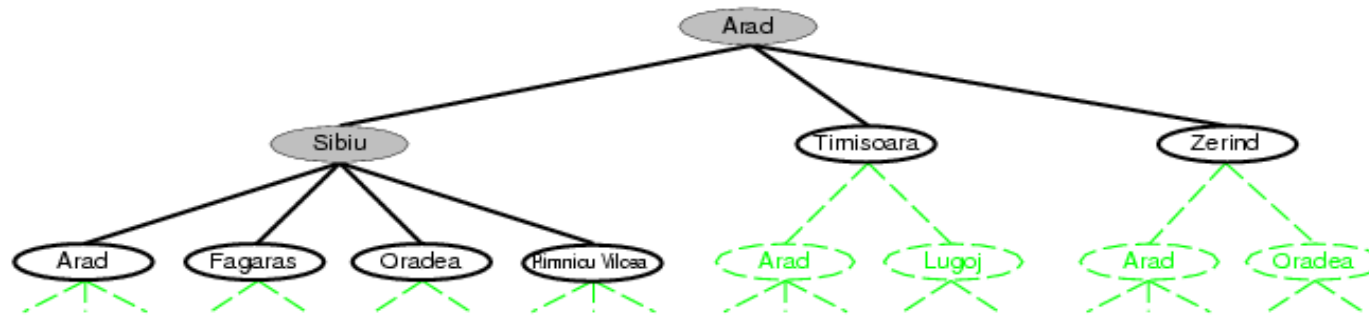
funcion BUSQUEDA-ARBOL(*problema, estrategia*) **return** una solucion o una falla
Inicializar árbol de Búsqueda al *estado inicial del problema*
do
 if no existen candidatos para expandir **then return** *falla*
 escoger nodo hoja para expandir de acuerdo a una *estrategia*
 if nodo contiene estado meta **then return** *solución*
 else expandir el nodo y agregar los nodos resultantes al árbol de Búsqueda
enddo

Ejemplo de árbol de Búsqueda



funcion BUSQUEDA-ARBOL(*problema, estrategia*) **return** una colucion o una falla
Inicializar árbol de Búsqueda al *estado inicial* del *problema*
do
 if no existen candidatos para expandir **then return** *falla*
 escoger nodo hoja para expandir de acuerdo a una estrategia
 if nodo contiene estado meta **then return** *solución*
 else expandir el nodo y agregar los nodos resultantes al árbol de Búsqueda
enddo

Ejemplo de árbol de Búsqueda



funcion BUSQUEDA-ARBOL(*problema, estrategia*) **return** una colucion o una falla

Inicializar árbol de Búsqueda al *estado inicial* del *problema*

do

if no existen candidatos para expandir **then return** *falla*

escoger nodo hoja para expandir de acuerdo a una estrategia

←Determina el proceso de Búsqueda!!

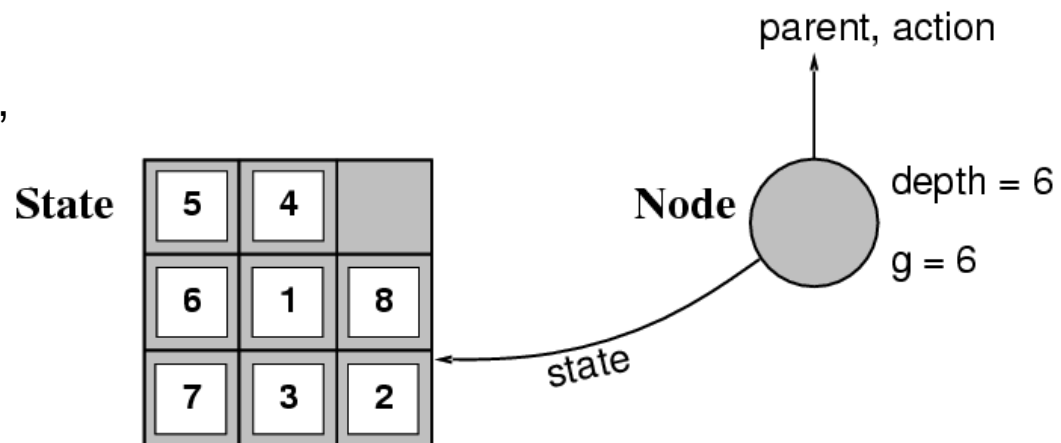
if nodo contiene estado meta **then return** *solución*

else expandir el nodo y agregar los nodos resultantes al árbol de Búsqueda

enddo

Implementación: estados vs. nodos

- Un **estado** es una (representación de) una configuración física
- Un **nodo** es una estructura de datos que constituye parte de un árbol de Búsqueda e incluye :
 - estado,
 - nodo padre,
 - acción,
 - costo del camino $g(x)$,
 - profundidad



- La función `expandir` crea nuevos nodos, llenando los demás campos y usando `funciónSucesor` del problema crea los correspondientes estados. □

Implementación: estados vs. nodos

Id nodo : identificador único del nodo ejemplo 1
Estado : estado actual ejemplo [7 ,2 ,4 ; 5, -1, 6;8 ,3 1]
nodoPadre : id del nodoPadre en este caso -1 pues es nodo raiz
Acción : en este caso nulo pues es nodo raiz
Costo : $g(x)=0$
Profundidad : = 0

Id nodo : 2
Estado : [7 ,-1 ,4 ; 5, 2, 6;8 ,3 1]
nodoPadre : 1
Acción : arriba
Costo : $g(x)=1$
Profundidad : 1

Id nodo : 3
Estado : [7 ,2 ,4 ; 5, 3, 6;8 ,-1 1]
nodoPadre : 1
Acción : abajo
Costo : $g(x)=1$
Profundidad : 1

Id nodo : 4
Estado : [7 ,2 ,4 ; -1,5, 6;8 ,3 1]
nodoPadre : 1
Acción : izquierda
Costo : $g(x)=1$
Profundidad : 1

Id nodo : 5
Estado : [7 ,2 ,4 ; 5, 6, -1;8 ,3 1]
nodoPadre : 1
Acción : derecha
Costo : $g(x)=1$
Profundidad : 1

Estrategias de Búsquedas

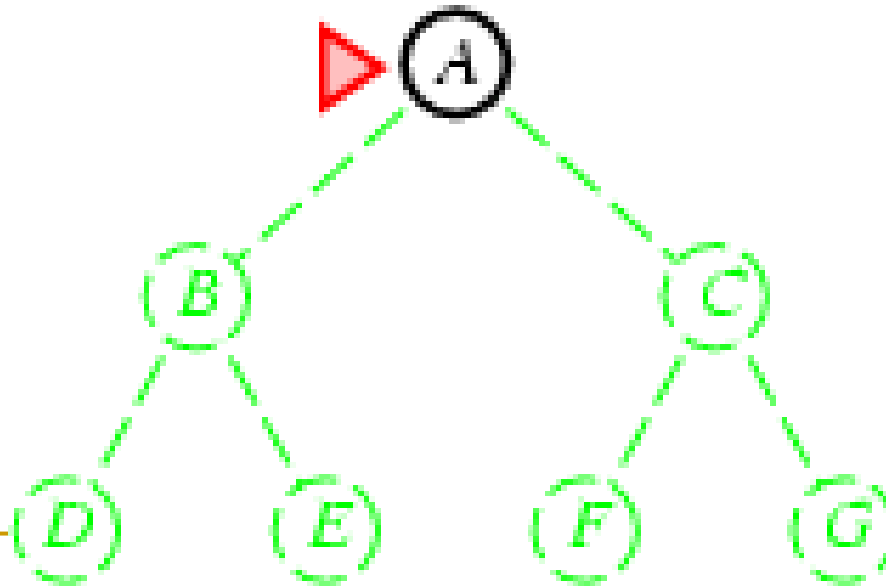
- Una estrategia de Búsqueda es definida escogiendo el **orden de la expansión de los nodos**
- Las estrategias son evaluadas por medio de las siguientes dimensiones:
 - **completitud**: encuentra siempre una solución si existe alguna?
 - **complejidad de tiempo**: número de nodos generados
 - **complejidad de espacio**: número máximo de nodos en memoria
 - **Optimalidad**: siempre halla la solución de menos costo? □
- Complejidad de tiempo y espacio son medidos en términos de
 - *b*: factor máximo de ramificación del árbol de Búsqueda
 - *d*: profundidad de la solución de menor costo
 - *m*: profundidad máxima del espacio de estados (puede ser ∞)

Estrategias de Búsqueda no informada

- Estrategias de Búsquedas **no informadas** usan solamente la información disponible en la definición del problema
- Búsqueda primero en amplitud
- Búsqueda de costo uniforme
- Búsqueda primero en profundidad
- Búsqueda en profundidad limitada
- Búsqueda de profundidad iterativa
- Búsqueda bidireccional

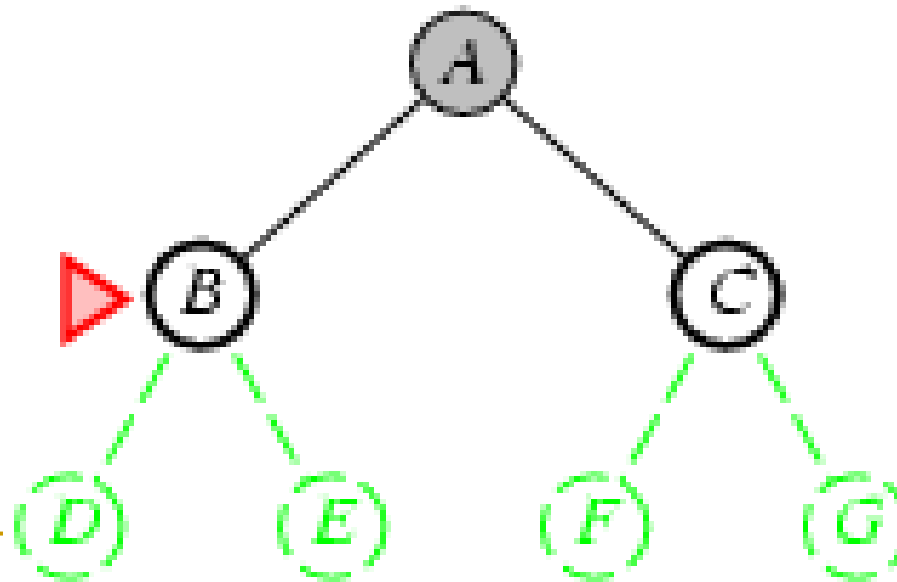
Búsqueda primero en amplitud

- Expandir el nodo mas interno sin visitar
- Implementación:
 - *nodos-borde* (nodos generados pero todavía no expandidos) es una cola FIFO



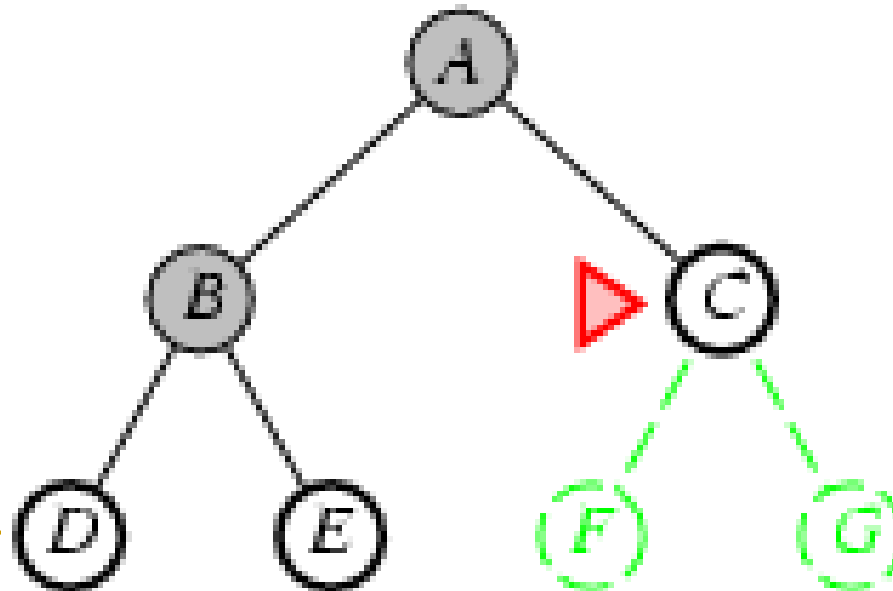
Búsqueda primero en amplitud

- Expandir el nodo mas interno sin visitar
- Implementación:
 - *nodos-borde* es una cola FIFO ejem, nuevos sucesores van al final



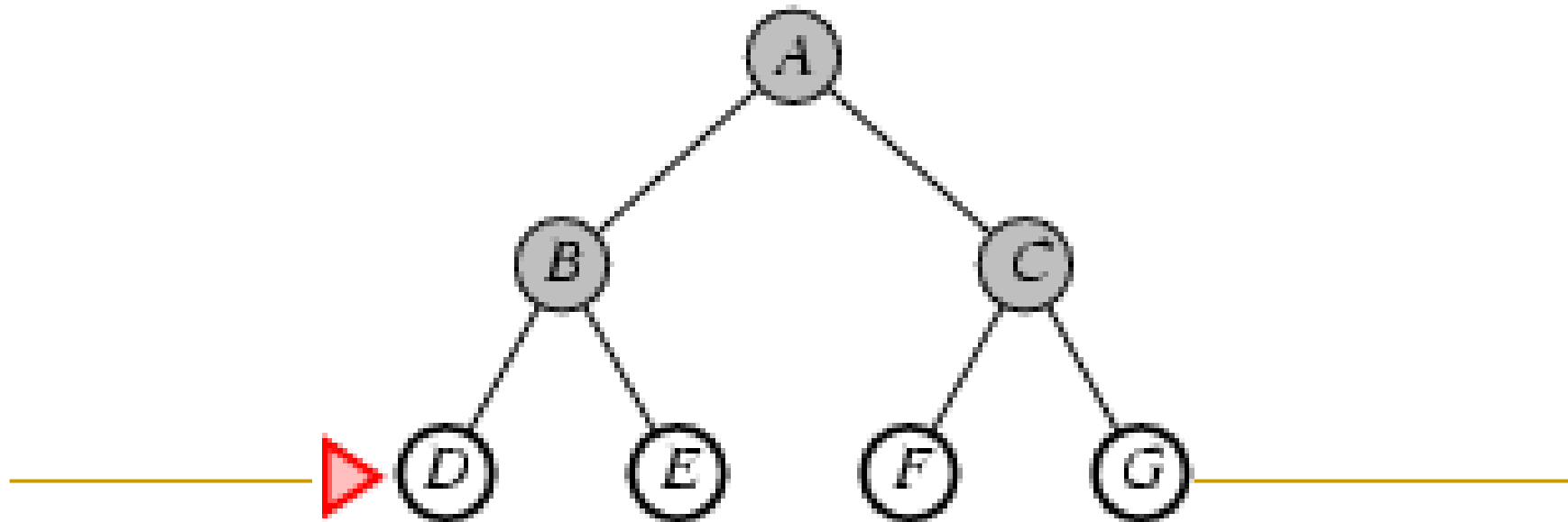
Búsqueda primero en amplitud

- Expandir el nodo mas interno sin visitar
- Implementación:
 - *nodos-borde* es una cola FIFO ejem, nuevos sucesores van al final



Búsqueda primero en amplitud

- Expandir el nodo mas interno sin visitar
- Implementación:
 - *nodos-borde* es una cola FIFO ejem, nuevos sucesores van al final



Propiedades de Búsqueda primero en amplitud

- Completo? Si (si b es finito)
- Tiempo? $1+b+b^2+b^3+\dots +b^d + b(b^d-1) = O(b^{d+1})$
- Espacio? $O(b^{d+1})$ (guarda cada nodo en memoria)
- Óptimo? Si (si costo = 1 por paso)

- **Espacio** es el problema mas grande (más que el Tiempo) □

Propiedades de la Búsqueda primero en amplitud

- Los requerimientos de memoria son el principal problema mas que el tiempo de ejecución
 - $b=10$,
 - procesamiento = 10000 nodos/segundo,
 - memoria = 1000 bytes/nodo

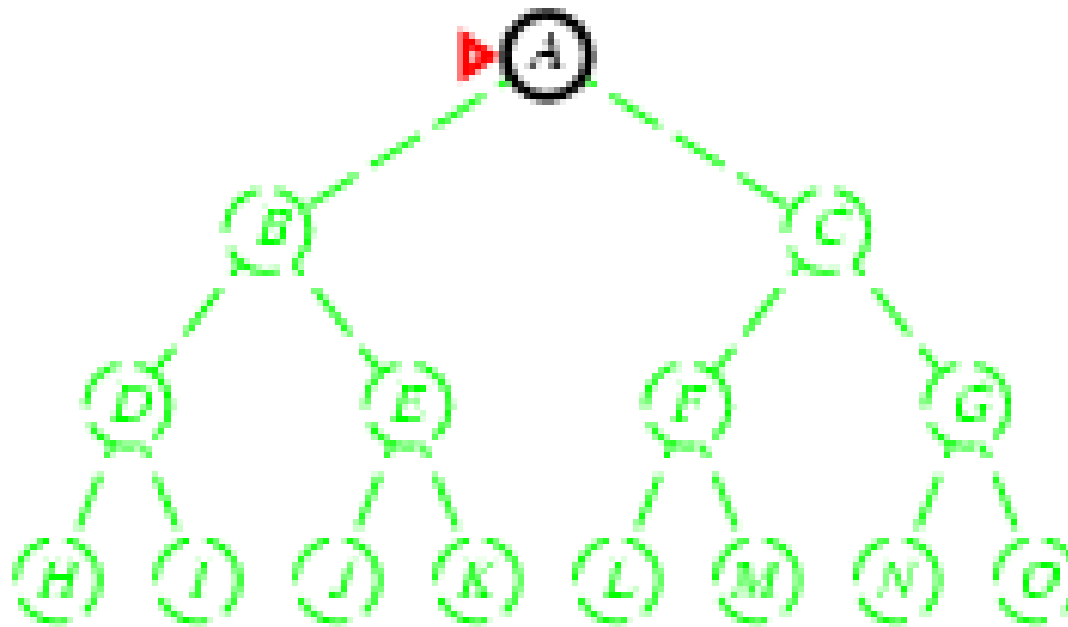
PROFUNDIDAD	NODOS	TIEMPO	MEMORIA
2	1100	0.11 segundos	1 megabyte
4	111100	11 segundos	106 megabytes
6	107	19 minutos	10 gigabytes
8	109	31 horas	1 terabyte
10	1011	129 dias	101 terabytes
12	1013	35 años	10 petabytes
14	1015	3523 años	1 exabyte

Búsqueda de costo uniforme

- Expande el nodo de menos costo sin expandir todavía □
- **Implementación:**
 - *nodos-borde* = cola ordenada por costo del camino
- Equivalente a primero por amplitud si los costos de los pasos son todos iguales
- Completo? Si, si el costo del paso es $\geq \epsilon$
- Tiempo?, $O(b^{1+(C^*/\epsilon)})$ donde C^* es el costo de la solución óptima
- Espacio? $O(b^{1+(C^*/\epsilon)})$
- Óptimo? Si – nodos expandidos en orden creciente de $g(n)$

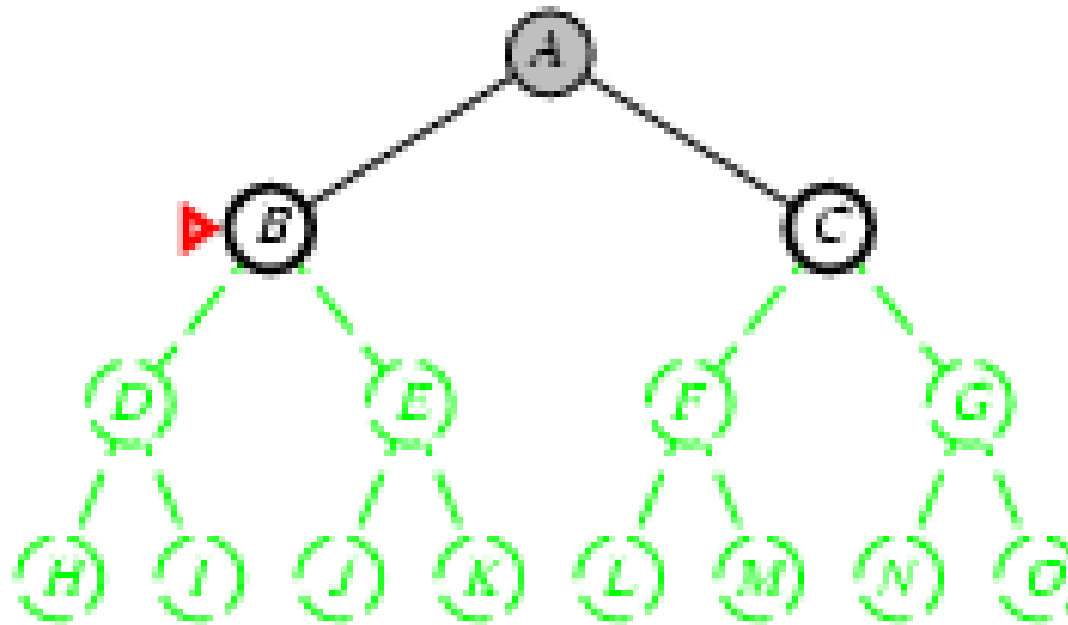
Búsqueda primero en profundidad

- Expande el nodo más profundo sin expandir todavía
- Implementación:
 - *nodos-borde* = cola LIFO ejem, colocar sucesores al frente



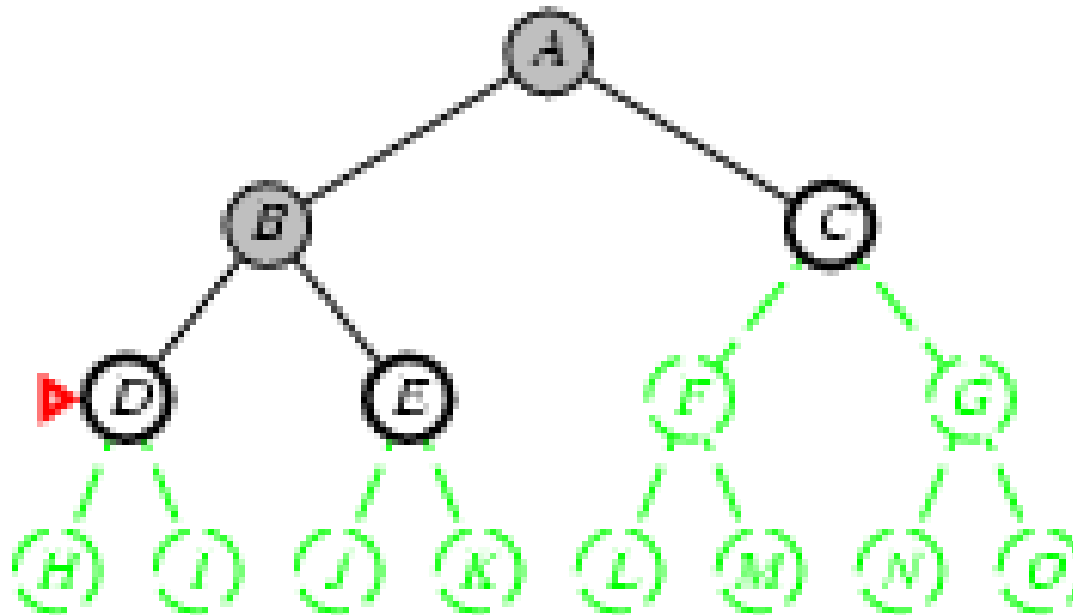
Búsqueda primero en profundidad

- Expande el nodo más profundo sin expandir todavía
- Implementación:
 - *nodos-borde* = cola LIFO ejem, colocar sucesores al frente



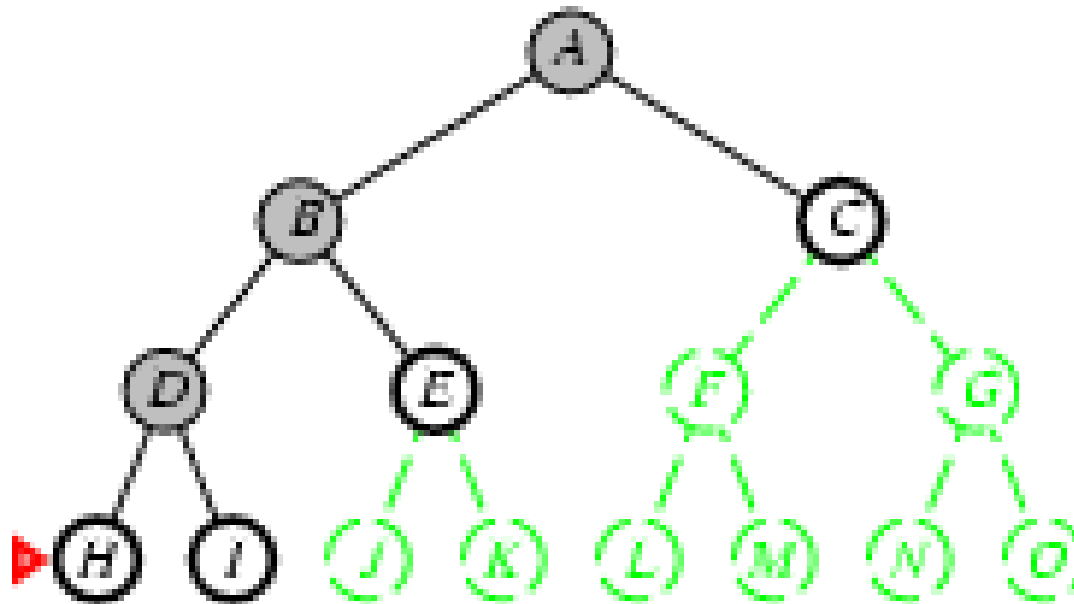
Búsqueda primero en profundidad

- Expande el nodo más profundo sin expandir todavía
- Implementación:
 - *nodos-borde* = cola LIFO ejem, colocar sucesores al frente



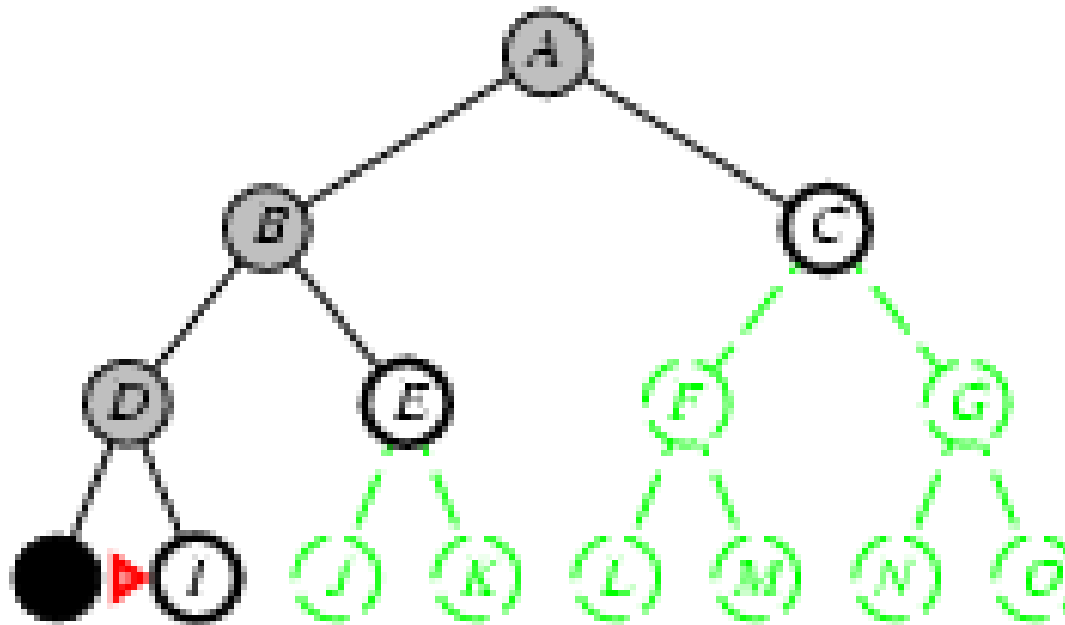
Búsqueda primero en profundidad

- Expande el nodo más profundo sin expandir todavía
- Implementación:
 - *nodos-borde* = cola LIFO ejem, colocar sucesores al frente



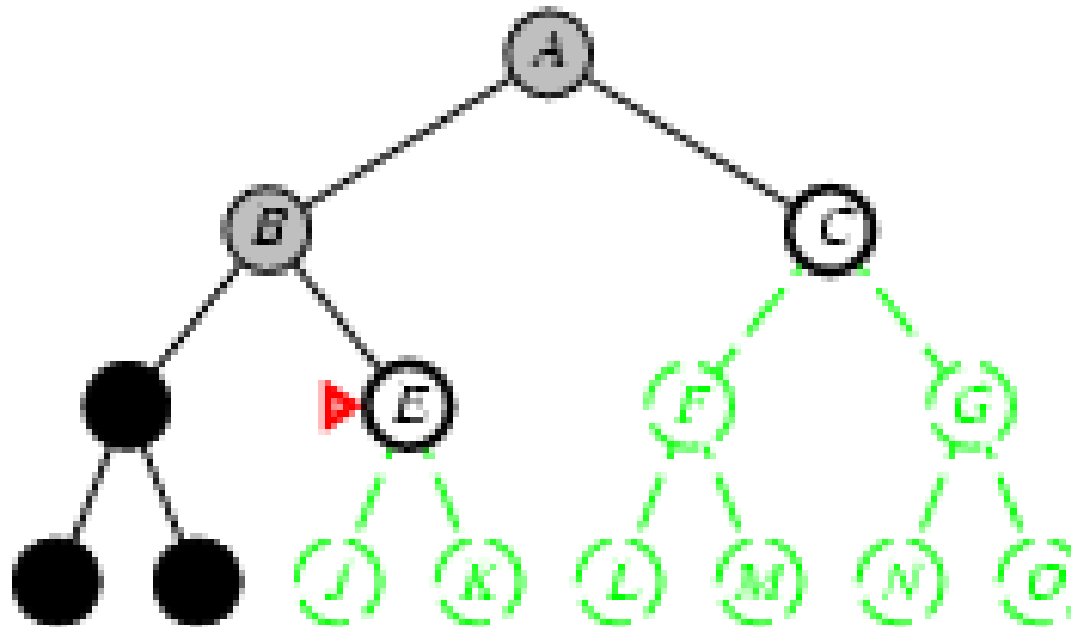
Búsqueda primero en profundidad

- Expande el nodo más profundo sin expandir todavía
- Implementación:
 - *nodos-borde* = cola LIFO ejem, colocar sucesores al frente



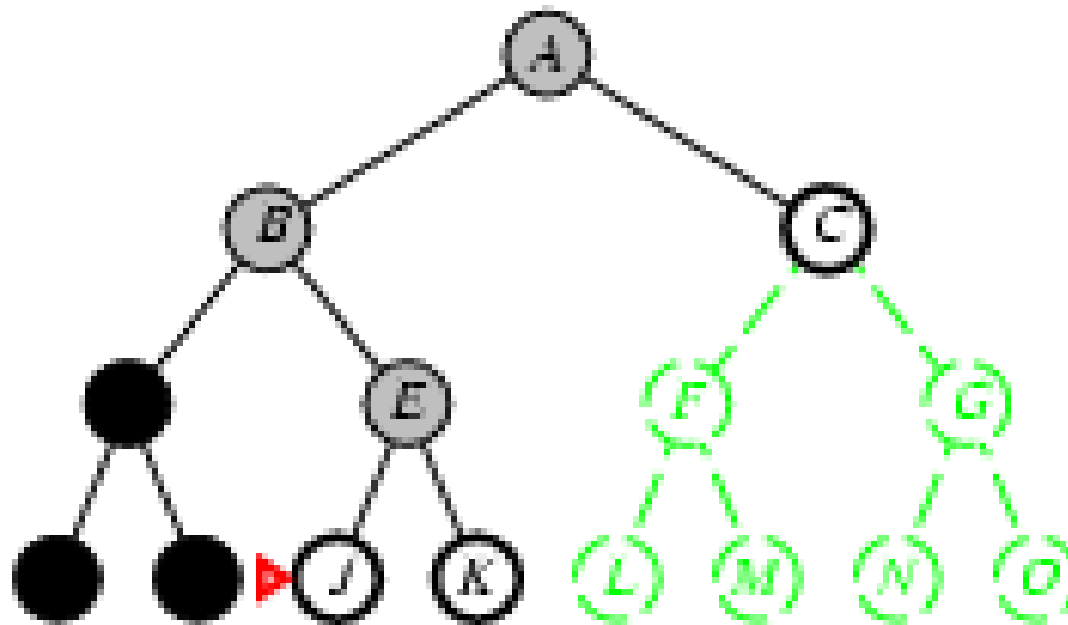
Búsqueda primero en profundidad

- Expande el nodo más profundo sin expandir todavía
- Implementación:
 - *nodos-borde* = cola LIFO ejem, colocar sucesores al frente



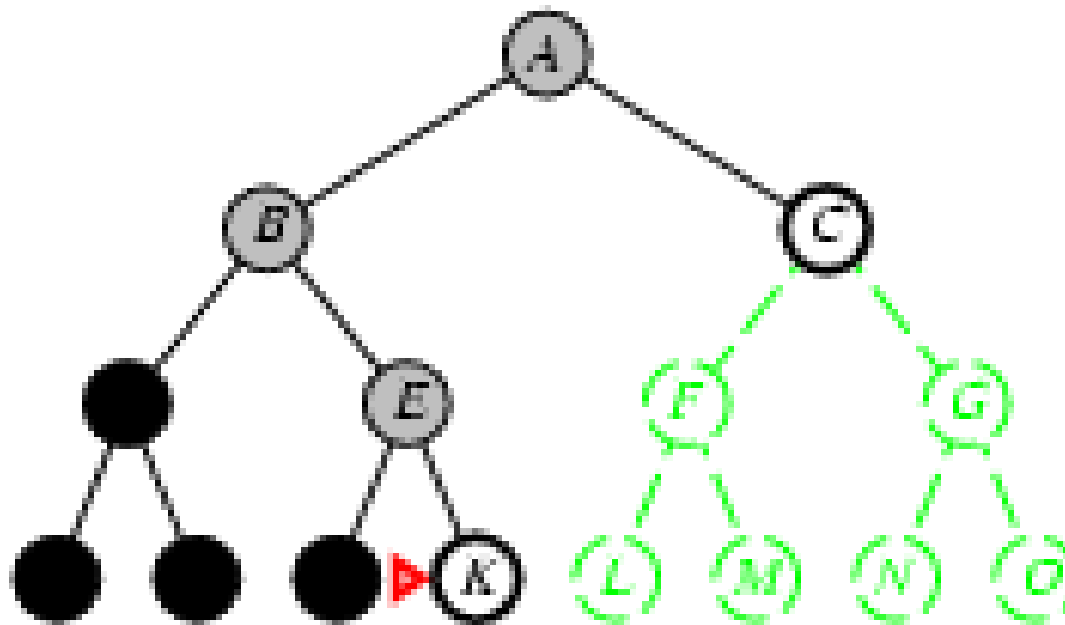
Búsqueda primero en profundidad

- Expande el nodo más profundo sin expandir todavía
- Implementación:
 - *nodos-borde* = cola LIFO ejem, colocar sucesores al frente



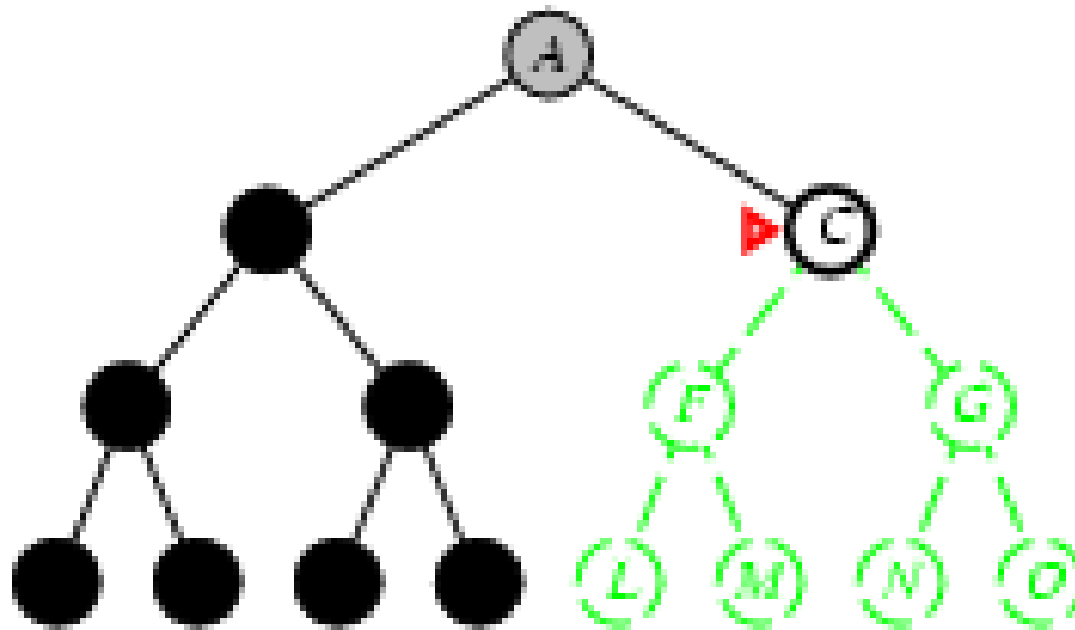
Búsqueda primero en profundidad

- Expande el nodo más profundo sin expandir todavía
- Implementación:
 - *nodos-borde* = cola LIFO ejem, colocar sucesores al frente



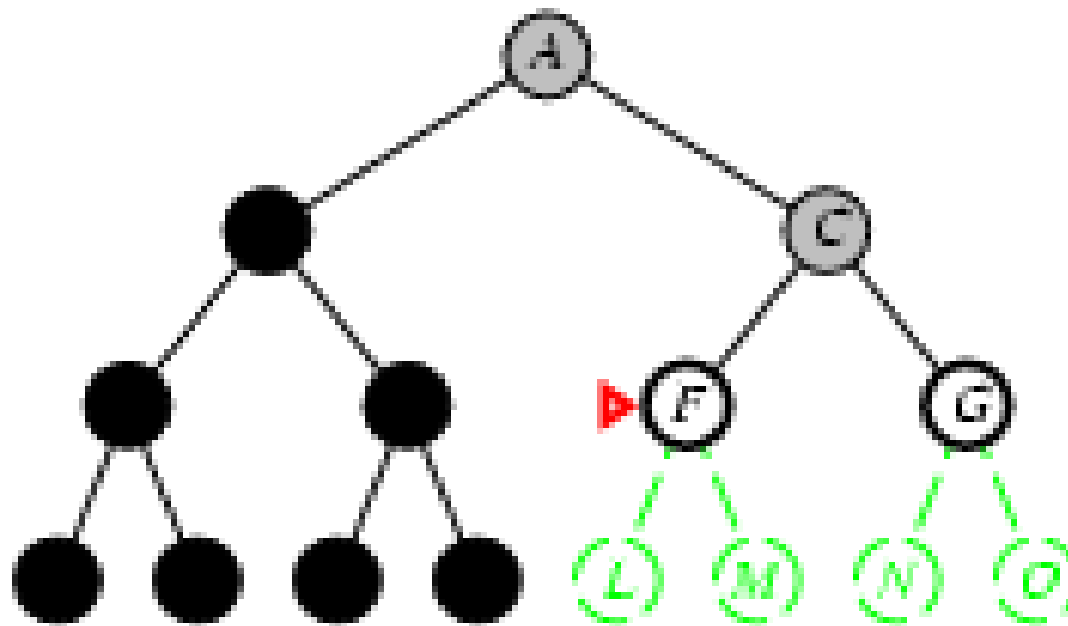
Búsqueda primero en profundidad

- Expande el nodo más profundo sin expandir todavía
- Implementación:
 - *nodos-borde* = cola LIFO ejem, colocar sucesores al frente



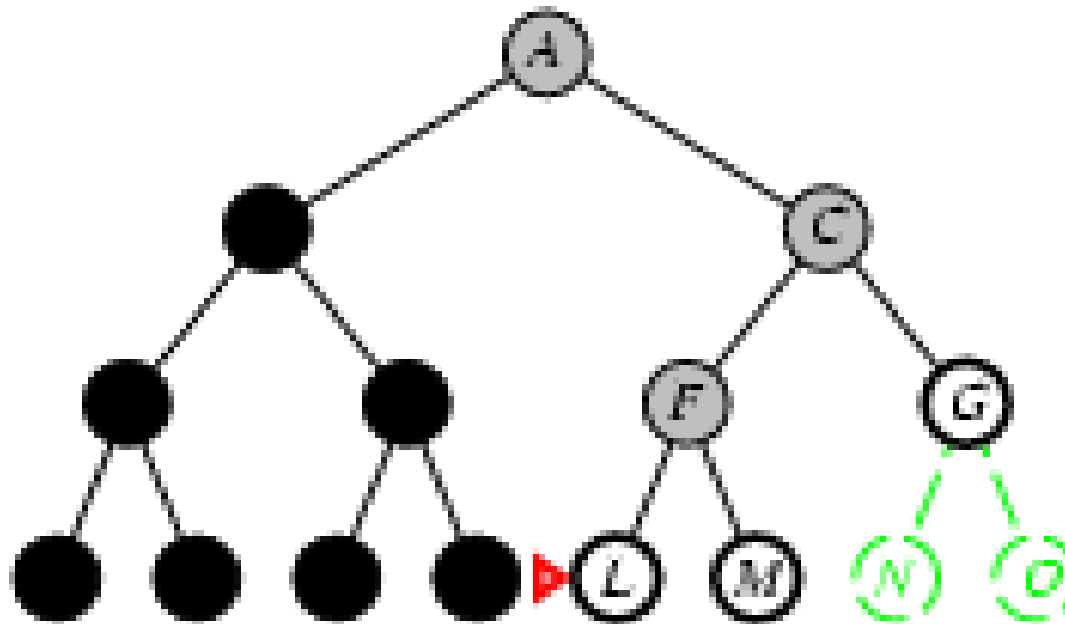
Búsqueda primero en profundidad

- Expande el nodo más profundo sin expandir todavía
- Implementación:
 - *nodos-borde* = cola LIFO ejem, colocar sucesores al frente



Búsqueda primero en profundidad

- Expande el nodo más profundo sin expandir todavía
- Implementación:
 - *nodos-borde* = cola LIFO ejem, colocar sucesores al frente



Propiedades de la Búsqueda primero en profundidad

- Completo? No: falla en espacios de profundidad infinita, espacios con loops
 - Modificar para evitar estados repetidos através del camino
→ Completo en espacios finitos
- Tiempo? $O(b^m)$: terrible si m es más grande que d
 - pero si las soluciones son densas, puede ser mucho más rápido que primero en amplitud □
- Espacio? $O(bm)$, Ejem., Espacio lineal!
- Óptimo? No

Busqueda en profundidad limitada

Búsqueda primero en profundidad con profundidad límite l , ejemplo. nodos a profundidad l no tienen sucesores.

- Conocimiento apriori del problema puede ser usado
- Soluciona el problema de hallar una ruta infinita.
- If $l < d$ resultados no satisfacen el criterio de completitud.
- If $l > d$ no es óptimo.

Búsqueda de profundidad iterativa

$l = 0$

Limit = 0



Búsqueda de profundidad iterativa

$l = 1$

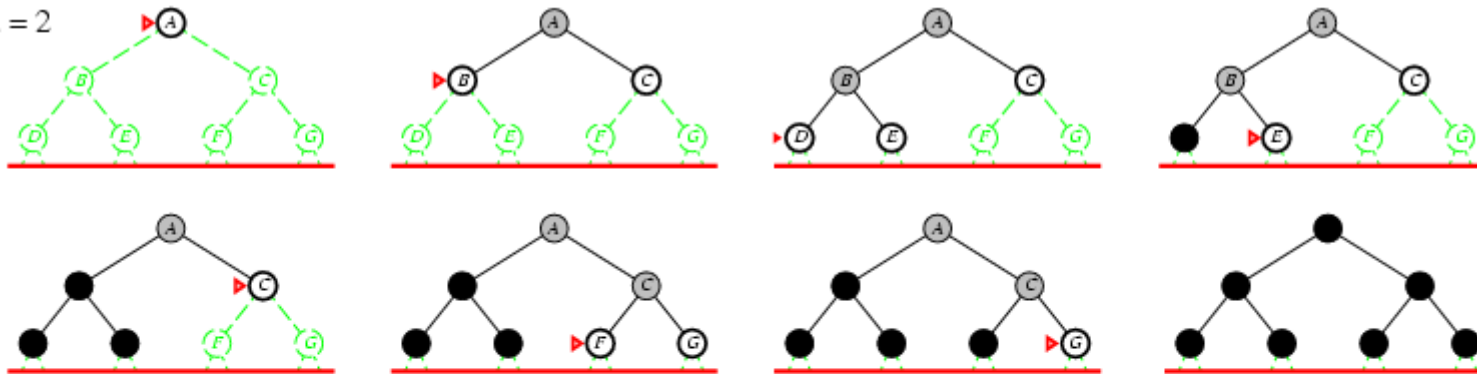
Limit = 1



Búsqueda de profundidad iterativa

$l=2$

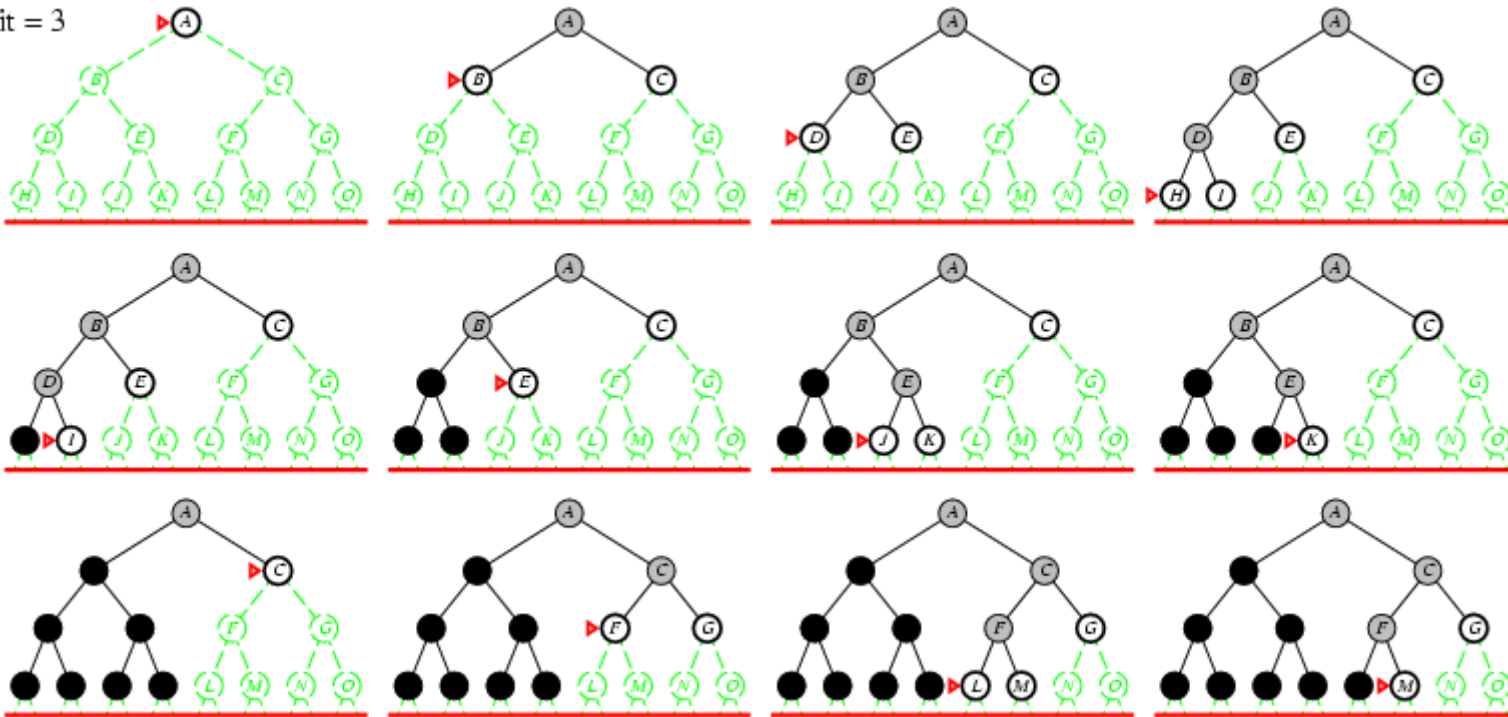
Limit = 2



Búsqueda de profundidad iterativa

$l=3$

Limit = 3



Búsqueda de profundidad iterativa

- Combina búsqueda primero en amplitud con búsqueda primero en profundidad
- Número de nodos generados en una búsqueda en profundidad limitada en la profundidad d con factor de b :

$$N_{DLS} = b^0 + b^1 + b^2 + \dots + b^{d-2} + b^{d-1} + b^d$$

- Número de nodos generados en una búsqueda de profundidad iterativa en la profundidad d con factor de ramificación b :

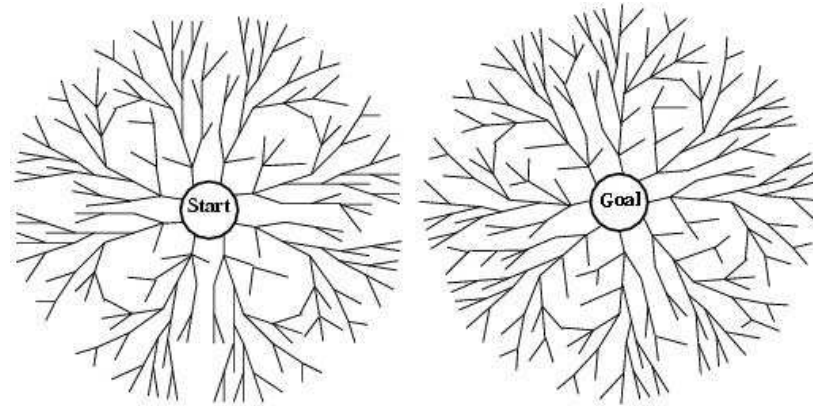
$$N_{IDS} = (d+1)b^0 + d b^1 + (d-1)b^2 + \dots + 3b^{d-2} + 2b^{d-1} + 1b^d$$

- Para $b = 10$, $d = 5$,
 - $N_{DLS} = 1 + 10 + 100 + 1,000 + 10,000 + 100,000 = 111,111$
 - $N_{IDS} = 6 + 50 + 400 + 3,000 + 20,000 + 100,000 = 123,456$
- Sobrecarga = $(123,456 - 111,111)/111,111 = 11\%$

Propiedades de la búsqueda de profundidad iterativa

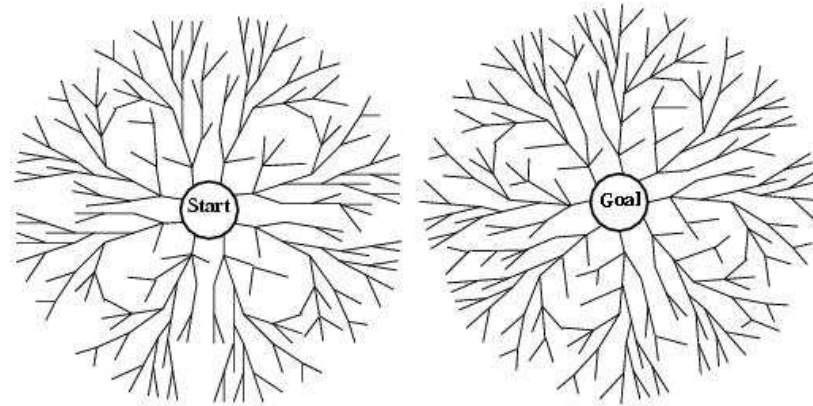
- Completo? Si
- Tiempo? $(d+1)b^0 + d b^1 + (d-1)b^2 + \dots + b^d = O(b^d)$
- Espacio? $O(bd)$
- Óptimo? Si, si costo de paso = 1

Búsqueda bidireccional



- Dos búsquedas simultaneas: nodo inicial y nodo meta
 - Motivación: $b^{d/2} + b^{d/2} \neq b^d$
- Verificar si el nodo a expandir es un nodo tipo nodos-borde (sin expandir) en la otra búsqueda.
- Complejidad de espacio es la mas significativa debilidad.
- La búsqueda es completa y óptima si ambas búsquedas son búsquedas por amplitud.

Búsqueda bidireccional



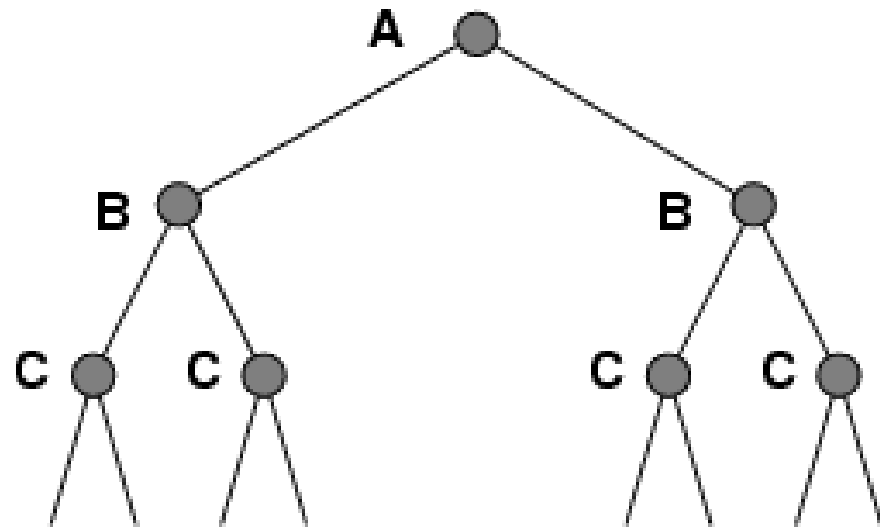
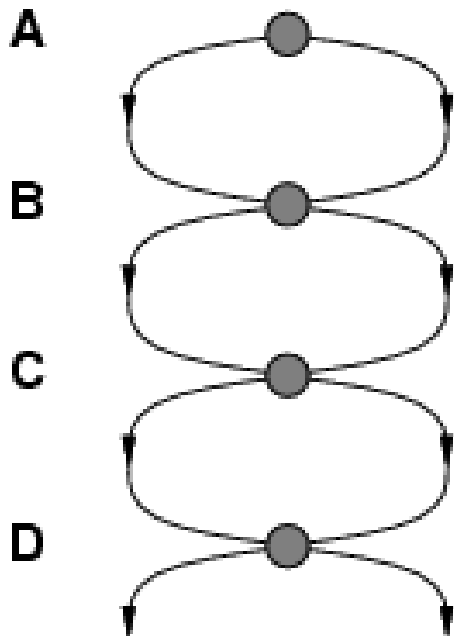
- El predecesor de cada nodo debe ser computado eficientemente cuando las acciones son fácilmente reversibles

Resumen de los algoritmos

Criterion	Primero-Amplitud	Costo-Uniforme	Primero-Profundidad	Profundidad-Limitada	Profundidad iterativa	Busqueda Bidireccional
Completo?	SI*	SI*	NO	SI, if $l \geq d$	SI	SI*
Tiempo	b^{d+1}	$b^{C*/e}$	b^m	b^l	b^d	$b^{d/2}$
Espacio	b^{d+1}	$b^{C*/e}$	b^m	b^l	b^d	$b^{d/2}$
Optimo?	SI*	SI*	NO	NO	SI	SI

Estados repetidos

- Si no se detectan los estados repetidos puede convertir un problema lineal en un exponencial!



Búsqueda en grafos

- Mantiene una lista de todos los nodos ya expandidos
- Idea usar tablas hash para almacenar nodos con estados ya generados (es suficiente guardar solamente el estado ya generado) y realizar búsquedas en tiempo $O(\alpha)$, donde α es el factor de carga de la tabla hash

Referencias Bibliográficas

- Capitulo 3 Artificial Intelligence: A Modern Approach, Russell and Norvig
- <http://www.aaai.org/AITopics/pmwiki/pmwiki.php/AITopics/Search>