



# IV. Métodos de Búsqueda

*Estudiaremos los métodos de búsqueda para resolver  
problema de la IA*

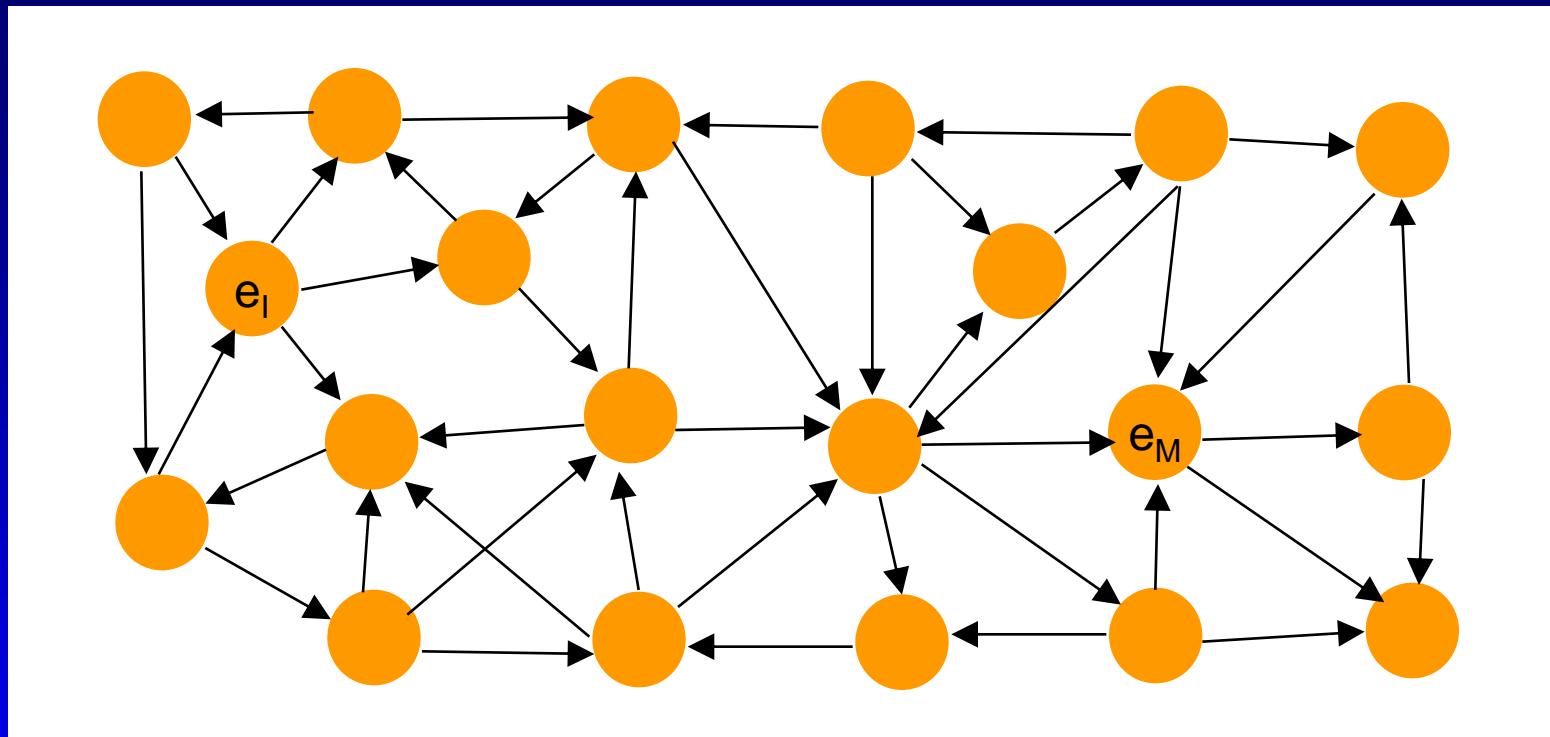
## 4. Métodos de Búsqueda

---

### Tópicos

- Métodos de búsqueda
- Árbol de estado
- Métodos a ciegas
- Función evaluadora
- Métodos informados

## 4.1 Métodos de búsquedas



### Problema de Búsqueda:

Determinar una secuencia válida de estados (reglas) que comience con el estado inicial ( $e_I$ ) y termine con el estado meta ( $e_M$ ), esto es, un camino  $e_I - e_M$

## 4.1 Métodos de búsquedas

---



¿Es posible aplicar los métodos de caminos mínimos?

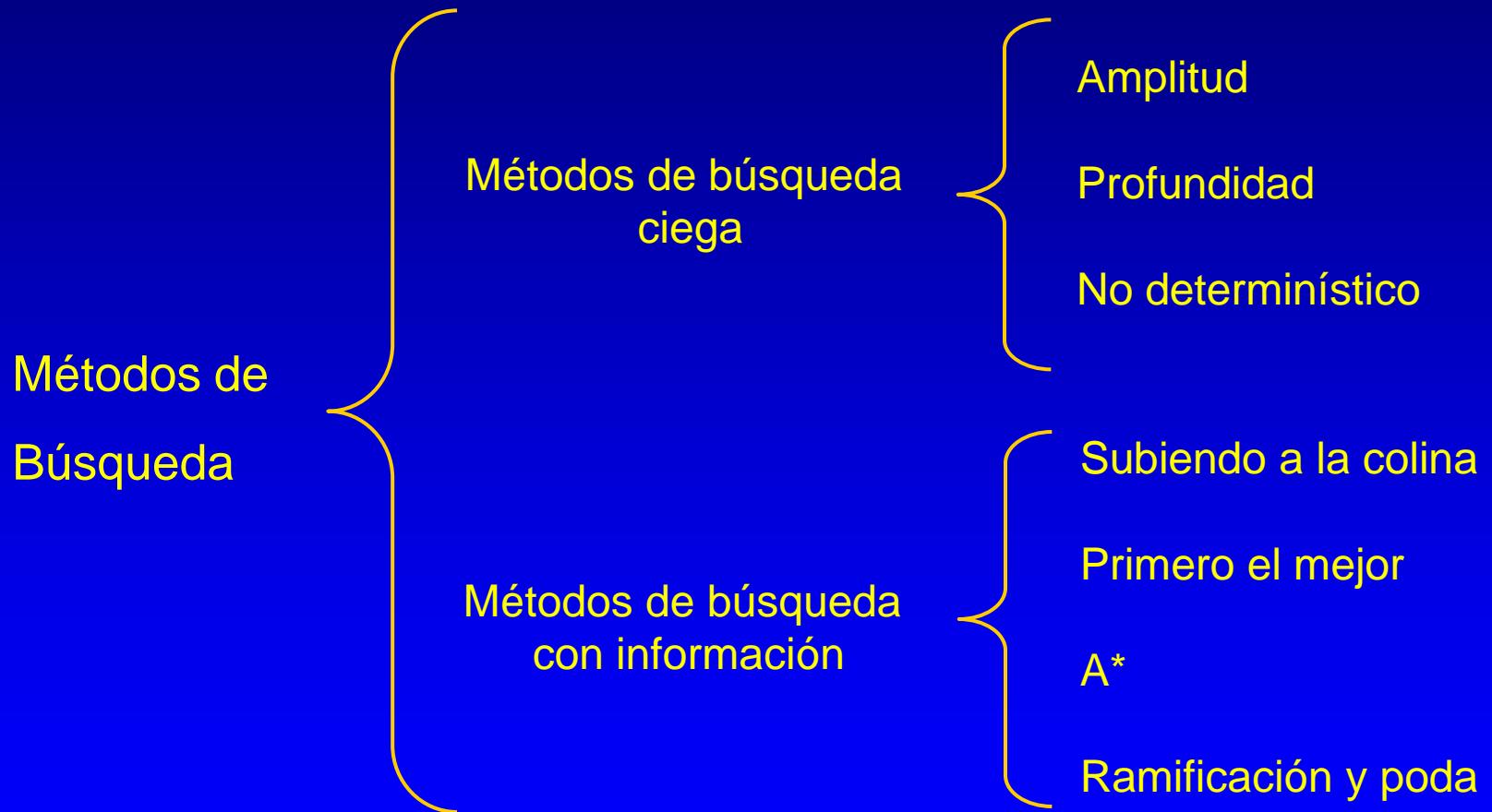
$O(n^2)$

La complejidad de los algoritmos de caminos mínimos es  $O(n^2)$  para un grafo de “ $n$ ” nodos (estados).

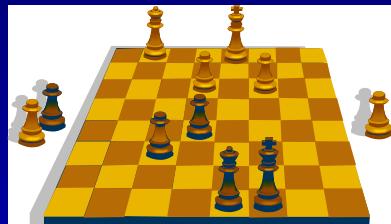
Infelizmente el espacio de estado es en general demasiado grande, lo que torna inviable en general el uso de estas técnicas para resolver problema de la IA. Así por ejemplo el problema del ajedrez presenta un espacio de estado de  $10^{40}$  nodos, esto es los algoritmos de caminos mínimos para resolver este problema requerirán una magnitud de  $10^{80}$  operaciones

## 4.1 Métodos de búsquedas

---



## 4.1 Métodos de búsquedas



Juega Negra

SP

Jugada A

Jugada B

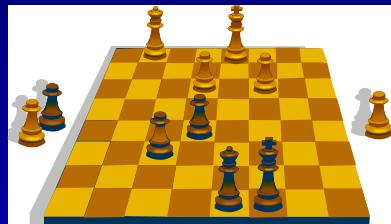
Jugada C

Jugada D

Jugada E

¿Qué jugada debe realizar la máquina?

## 4.1 Métodos de búsquedas



Juega Negra

SP

**Jugada A**

Pn come Pb

**Jugada B**

Mueve Cn

**Jugada C**

An come Reina

**Jugada D**

Pn come Tn

**Jugada E**

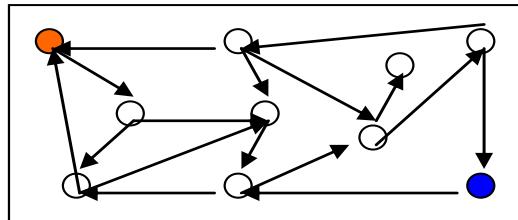
Cn come Cb

¿Qué jugada debe realizar la máquina?

## 4.2 Conceptos: Árbol de Estado

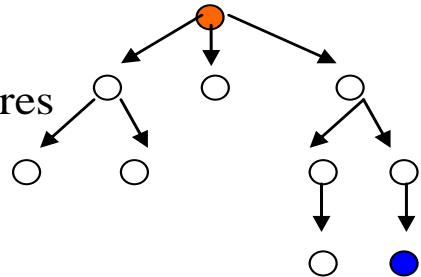
Los métodos de búsqueda se basan en el árbol de estado

Espacio de estados



Raíz

Sucesores



- Estado Meta
- Estado Inicial

Árbol de Estado

## 4.2 Conceptos: Árbol de Estado

---

### Como se construye un árbol de estado

#### Ejemplo: Vasijas de Agua

Ud. tiene dos vasijas vacías de 4 y 3 litros, y sin marcas.  
Llene exactamente 1 litro de agua en la vasija de 4.  
Considere que existe una bomba de agua.

#### Solución

Estado Inicial: (0,0)

Estado Meta: (1,0), (1,1), (1,2), (1,3)

## 4.2 Conceptos: Árbol de Estado



### Como se construye un árbol de estado

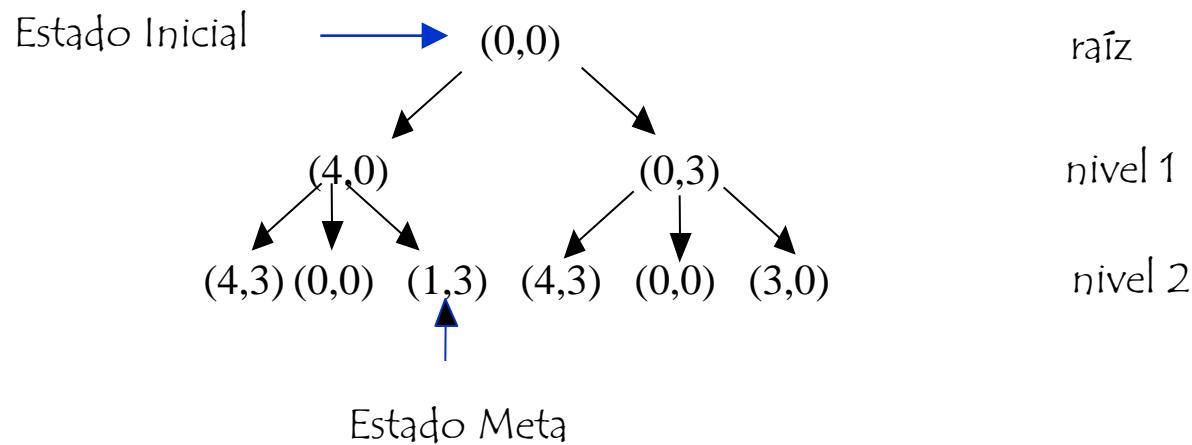
Regla	Condición	Nuevo Estado
Llene_4	$x < 4$	(4, y)
Llene_3	$y < 3$	(x, 3)
Vacía_4	$x > 0$	(0, y)
Vacíe_3	$y > 0$	(x, 0)
Pasa_43	$0 < x$ $y < 3$	( $x - m, y + m$ )
Pase_34	$y > 0$ $x < 4$	( $x + n, y - n$ )

donde:  $m = \min\{x, 3-y\}$ ,  $n = \min\{y, 4-x\}$

Reglas para el Problema de las Vasijas de Agua

## 4.2 Conceptos: Árbol de Estado

### Como se construye un árbol de estado



Árbol de Estado – Problema de las Vasijas

## 4.3 Métodos ciegos (sin información)

---

### Métodos de Búsqueda Ciega

Los métodos ciegos son procedimiento sistemáticos de búsqueda del estado meta en el árbol de estado.

Son llamados de métodos ciegos, porque usan estrategias de búsqueda que solo consideran la relación de precedencia entre estados. La información sobre el beneficio, utilidad, lucro de pasar de un estado para otro estado no es considerado.

Los métodos de búsqueda ciega más conocidos son:

- **Búsqueda en amplitud**
- **Búsqueda en profundidad**
- **Búsqueda no determinística (aleatorio)**

## 4.3.1 Búsqueda en Amplitud

---

### Procedimiento

Inicie en el nodo raíz del árbol de estado con el estado inicial. Si el nodo corresponde al estado meta entonces termine, caso contrario genere los nodos sucesores no redundante del presente nivel (use el sistema de producción). Si alguno de los nodos del primer nivel corresponde al estado meta entonces termine, caso contrario genere los nodos sucesores no redundante del primer nivel, esto es los nodos del segundo nivel (use el sistema de producción). El proceso se repite hasta encontrar el estado meta o cuando no sea posible generar nuevos sucesores.

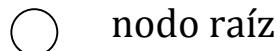
## 4.3.1 Búsqueda en Amplitud



### Procedimiento

#### Iteración 0:

Nivel 0



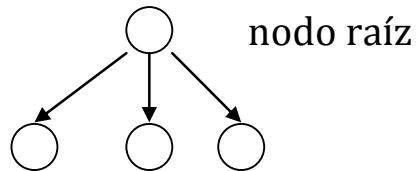
nodo raíz

Test: verificar si es estado meta.

Si la rpta es SI → termina, de lo contrario se genera los sucesores no redundantes.

#### Iteración 1:

Nivel 0



nodo raíz

Nivel 1

Test: verificar si algún nodo del nivel 1 es estado meta.

Si la rpta es SI → termina, de lo contrario se genera los sucesores no redundantes del nivel 1.

## 4.3.1 Búsqueda en Amplitud

### Procedimiento

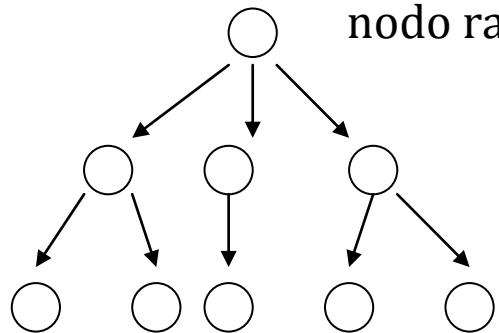
Iteración 2:

nivel 0

nodo raíz

nivel 1

nivel 2



Test: verificar si algún estado del nivel 2 es estado meta

Si la rpta es SI → termina, de lo contrario

Se genera los sucesores no redundantes del nivel 2.

## 4.31 Búsqueda en Amplitud

---

### Implementación – Listas

**LE:**

Lista de nodos en espera de proceso (nodos no comparados con el estado meta)

**LV:**

Lista de nodos ya procesado (nodos ya comparados con el estado meta)

## 4.3.1 Búsqueda en Amplitud

---

### Implementación – Listas

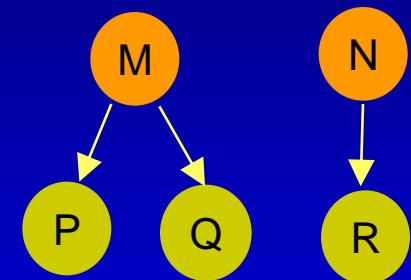
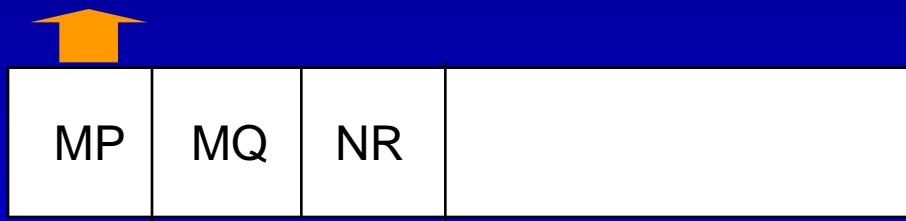
*Con el fin de construir la solución (secuencia de estados desde el estado inicial hasta el estado meta) registraremos en LE y LV cada nodo (estado) con su antecesor.*

## 4.3.1 Búsqueda en Amplitud

### Implementación – Listas

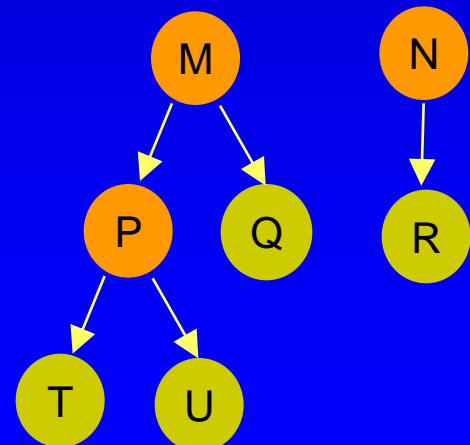
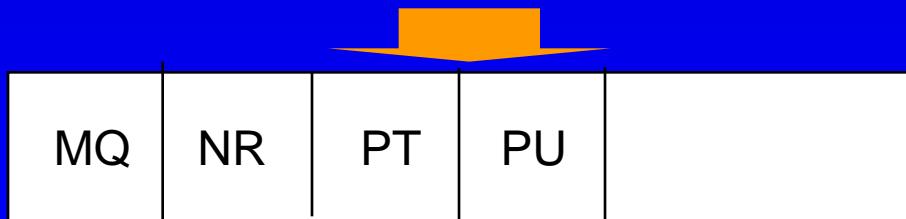
LE:

**PROCESAR**



LE:

**REGISTRO HIJOS(P)**



## 4.3.1 Búsqueda en Amplitud

### Algoritmo - Listas

#### Test de Parada

##### Condición de éxito

El estado a procesar es meta

(F,P):= Primer(LE)

Si ( P es Meta ) entonces PARE;

##### Condición de Fracaso

No es posible seguir buscando

Si ( LE = () ) entonces

Escribir("no hay solución"), PARE

## 4.3.1 Búsqueda en Amplitud

### Algoritmo - Listas

#### Inicio

1. LE := ((O,Estado\_Inicial)); LV:=();

#### Test de Parada

2. Si ( LE = () ) entonces

*Escribir(“no hay solución”), PARE;*

3. (F,P):= Primer(LE)

4. Si ( P es Meta ) entonces

*Determinar\_Solucion, PARE;*

#### Genera Sucesores:

5. Adiciona\_ultimo((F,P), LV);

6. Elimina\_primer(LE);

7. Para (Nodo ∈ (Hijos(P) - LV))

Adicionar\_ultimo((P,Nodo), LE);

8. Ir a 2

## 4.3.1 Búsqueda en Amplitud

### Algoritmo - Listas

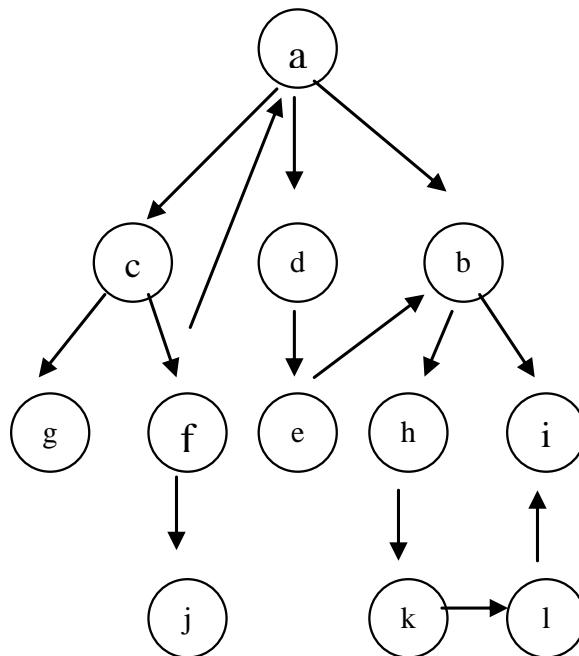
#### Determinar\_Solucion

Para determinar la solución, esto es el camino de estados que inicia en el estado inicial y termina en el estado meta, basta concatenar al estado meta su antecesor, y a este su antecesor y así sucesivamente hasta alcanzar el estado inicial.

La búsqueda de los antecesores a los estados se debe hacer desde LE y LV cuando se ha encontrado el estado meta.

## 4.3.1 Búsqueda en Amplitud

### Algoritmo - Listas



Ejemplo: Determine un Camino c-i  
Considere la lectura en sentido horario

## 4.3.2 Búsqueda en Profundidad

---



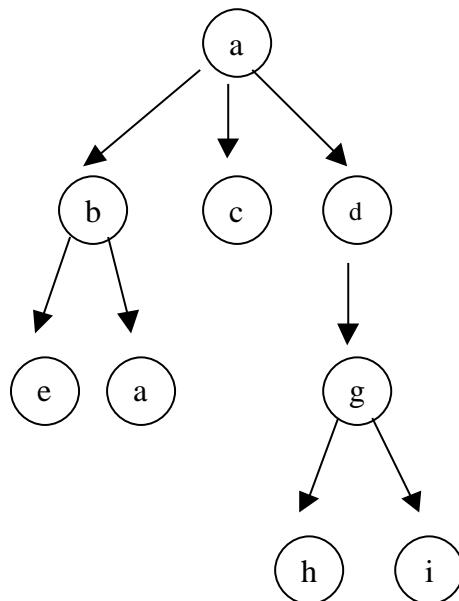
### Conceptos

- Una hoja de un árbol es un nodo del árbol que no tiene sucesores.
- Una rama de un árbol es un camino que inicia en el nodo raíz y termina en un nodo hoja.

## 4.3.2 Búsqueda en Profundidad

### Conceptos: Ramas

Ejemplo. Determine las ramas del siguiente árbol



Las ramas son: a, b, e      a, b, a      a, c      a, d, g, h      a, d, g, i.

## 4.3.2 Búsqueda en Profundidad

---

### Procedimiento

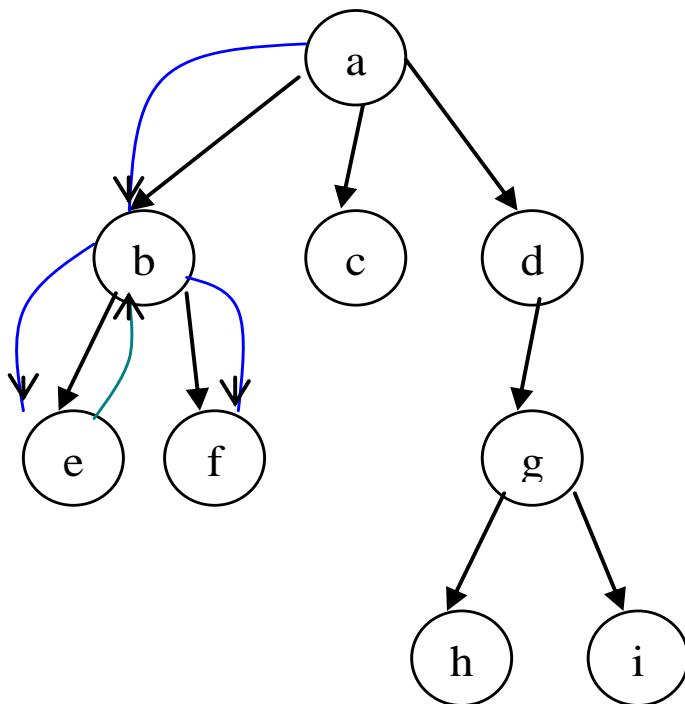
El método consiste en una búsqueda por las ramas del árbol de estado.

Si en una de las ramas se encuentra el estado meta entonces el procedimiento termina, de lo contrario se pasa a investigar sobre otra rama no redundante. El procedimiento se repite hasta encontrar el estado meta (éxito) o hasta que no existan más ramas a investigar (fracaso).

## 4.3.2 Búsqueda en Profundidad



### Procedimiento



## 4.3.2 Búsqueda en Profundidad

---

### Implementación – Listas

**LE:**

Lista de nodos en espera de proceso (nodos no comparados con el estado meta)

**LV:**

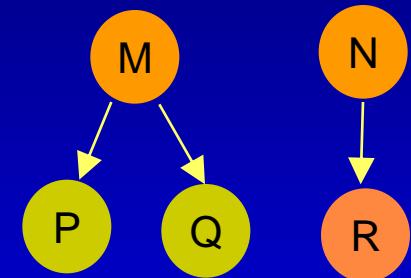
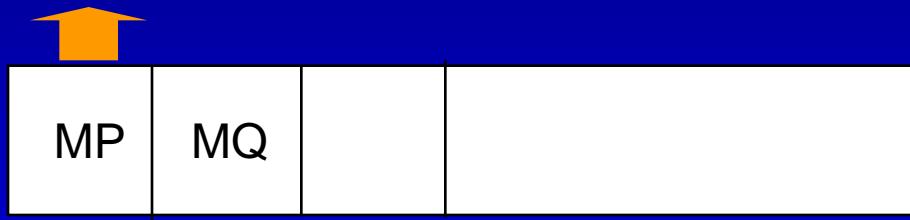
Lista de nodos ya procesado (nodos comparados con el estado meta)

## 4.3.2 Búsqueda en Profundidad

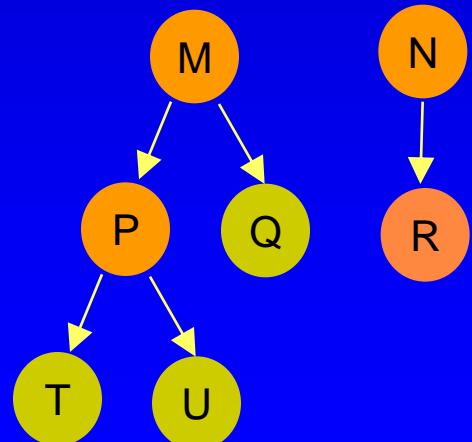
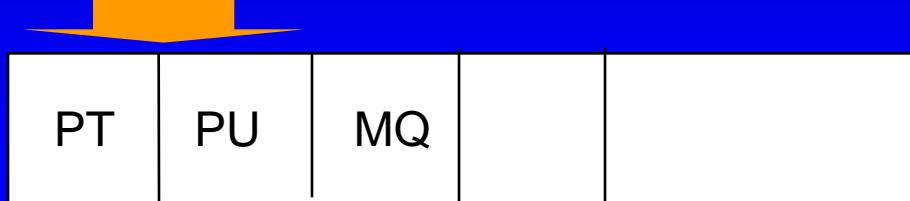
### Implementación – Listas

LE:

PROCESAR



LE: REGISTRO HIJOS(P)



## 4.3.2 Búsqueda en Profundidad

### Algoritmo - Listas

#### Inicio

1. LE := ((O,Estado\_Inicial)); LV:=();

#### Test de Parada

2. Si ( LE = () ) entonces

*Escribir(“no hay solución”), PARE;*

3. (F,P):= Primer(LE)

4. Si ( P es Meta ) entonces

*Determinar\_Solucion, PARE;*

#### Genera Sucesores:

5. Adiciona\_ultimo((F,P), LV);

6. Elimina\_primer(LE);

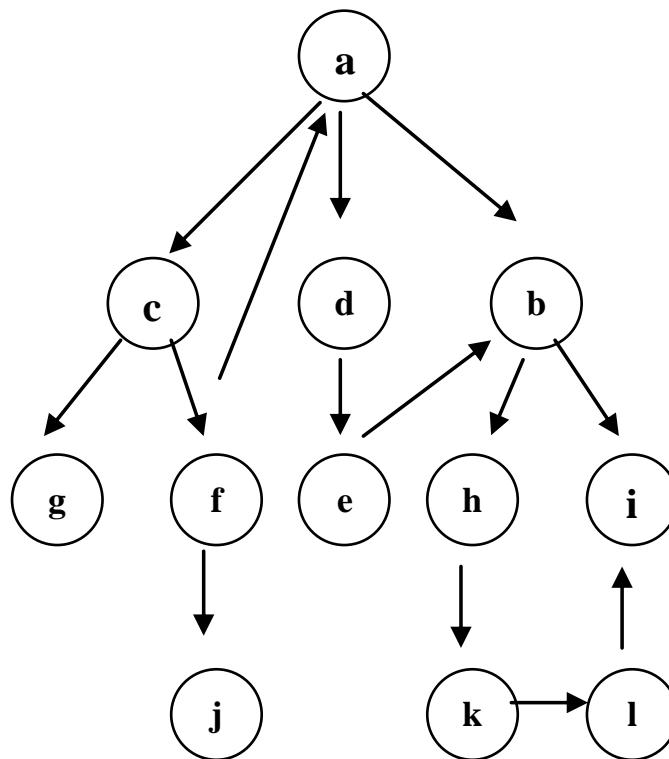
7. Para (Nodo ∈ (Hijos(P) - LV))

*Adicionar\_primero((P,Nodo), LE);*

8. Ir a 2

## 4.3.2 Búsqueda en Profundidad

Ejemplo: Determine un camino c-i.  
Considere la lectura en sentido horario



## 4.3.3 Búsqueda No Determinística

---



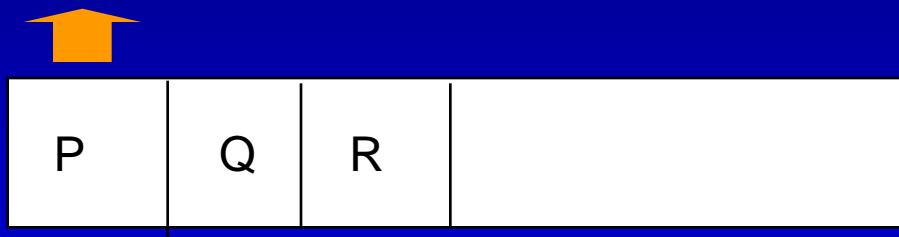
### Observaciones: Amplitud - Profundidad

- El estado a ser procesado en ambos métodos es el primero de la lista LE.
- Las listas generadas en los métodos de búsqueda en amplitud y en profundidad son registradas respectivamente al final e inicio de LE.

## 4.3.3 Búsqueda No Determinística

LE:

PROCESO



REGISTRO



Profundidad

Amplitud

¿Es el primer nodo el mejor para ser procesado?

## 4.3.3 Búsqueda No Determinística

---

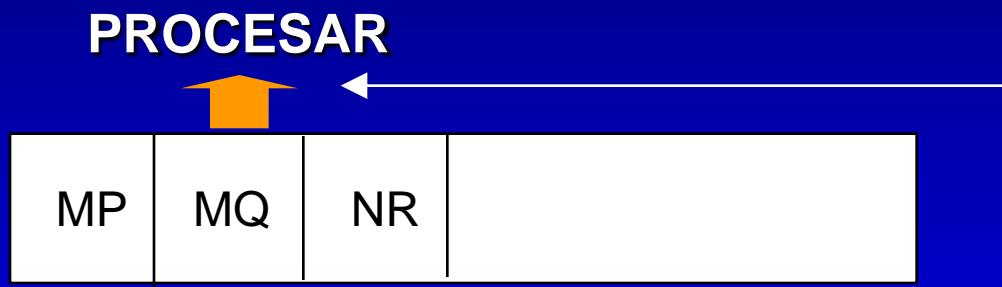
### Procedimiento

**En este método, el nodo a procesar es seleccionado aleatoriamente de la lista LE, los nodos sucesores son colocados al inicio o final de LE.**

## 4.3.3 Búsqueda No Determinística

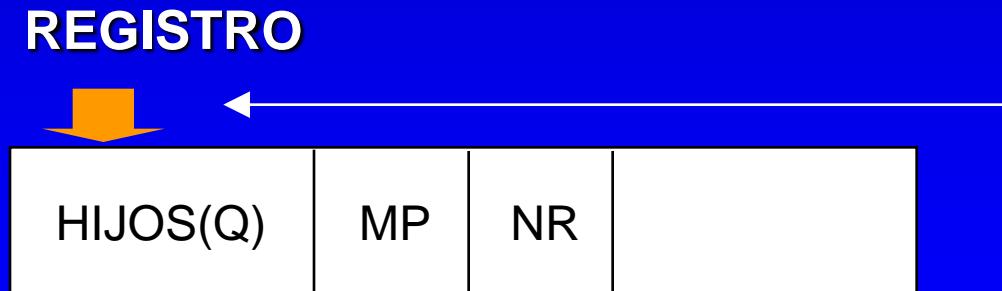
### Implementación – Listas

LE:



Selección aleatoria

LE:



Conveniencia

## 4.3.3 Búsqueda No Determinística

### Algoritmo - Listas

#### Inicio

1. LE := ((O,Estado\_Inicial)); LV:=();

#### Test de Parada

2. Si ( LE = () ) entonces

*Escribir(“no hay solución”), PARE;*

3. (F,P):= *Aleatorio*(LE)

4. Si ( P es Meta ) entonces

*Determinar\_Solucion, PARE;*

#### Genera Sucesores:

5. *Adiciona\_ultimo*((F,P), LV);

6. *Elimina\_aleatorio*(LE);

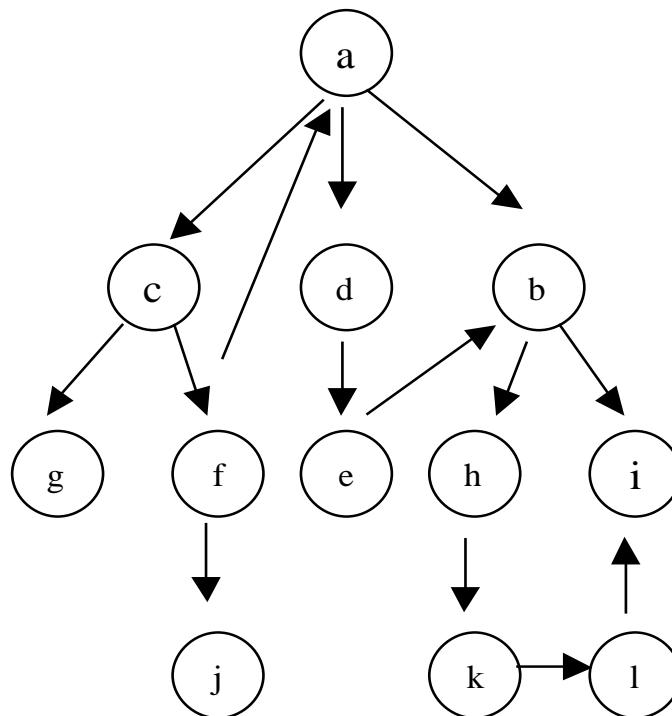
7. Para (Nodo  $\in$  (Hijos(P) - LV))

*Adicionar\_primer((P,Nodo), LE);*

8. Ir a 2

## 4.3.3 Búsqueda No Determinística

Ejemplo: Determine un camino c-i.  
Consideré la lectura en sentido horario



## 4.3. Métodos de Búsqueda a Ciegas - Observaciones

---



- La complejidad de los métodos ciegos es no polinomial.
- Si se conoce a priori que el estado meta está próximo (muy distante) al estado inicial es adecuado usar el método de búsqueda en amplitud (profundidad).

## 4.4 Función Evaluadora (Mérito)

---

### Definición

Sea  $\Pi$  un problema de inteligencia artificial y  $E$  su espacio de estado, una función que asocia a cada estado de  $E$  un número real, esto es:

$$f: E \rightarrow R$$

es llamado de función de mérito o función evaluadora asociada a  $\Pi$ .

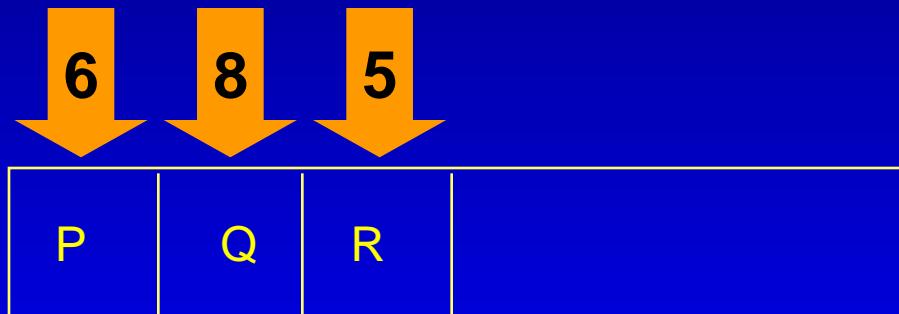
La función de mérito mide la utilidad de la información asociado al estado y puede significar riesgo, utilidad, costo, rentabilidad, distancia, ganancia, etc.

## 4.4 Función Evaluadora (Mérito)

---

### Ejemplo

LE:



¿Cuál es el mejor nodo para ser procesado?

## 4.4 Función Evaluadora (Mérito)

---



### Ejemplo

#### Problema

Juego

Agente viajero

Tres en raya

Vasija de Agua

Selección de Proy.

#### Función de mérito

Utilidad asociada a cada posible movimiento

Distancia asociada a cada posible ciudad a visitar

Mayor número de X colineales (caso juega X) donde no hay O

Litros faltantes para alcanzar la solución

Utilidad asociada a cada proyecto

## 4.5 Métodos Informados (con información)

---



**Los métodos informados o con información son procedimiento sistemáticos de búsqueda del estado meta en el árbol de estado, que usan la información asociada a los estados para una mejor decisión en el proceso de búsqueda. Para evaluar la información de los estados considera la función evaluadora.**

**La información sobre el beneficio, utilidad, lucro de pasar de un estado para otro estado si es considerado en el proceso de búsqueda.**

**Los métodos de búsqueda informados más conocidos son:**

- **Subiendo a la colina**
- **Primero el mejor**
- **A\***
- **Ramificación y poda**

## 4.5.1 Método Subiendo a la Colina

---



### Procedimiento

El procedimiento es semejante a la búsqueda en profundidad con la diferencia que los nodos sucesores son ordenados del mejor al peor valor de su función mérito antes de adicionarse a la lista LE. Esto es, el nodo a ser procesado será el “mejor” nodo sucesor, de acuerdo a la función de mérito.

## 4.5.1 Método Subiendo a la Colina

### Implementación – Listas

LE:

**PROCESAR**



LE:

**REGISTRO**



## 4.5.1 Método Subiendo a la Colina

---

### Implementación – Listas

La ordenación de los sucesores puede ser:

ordenación:

de menor a mayor

de mayor a menor

función evaluadora

distancia, costo, número de pasos

utilidad, ganancia, beneficios

## 4.5.1 Método Subiendo a la Colina

### Algoritmo - Listas

#### Inicio

1. LE := ((O,Estado\_Inicial)); LV:=();

#### Test de Parada

2. Si ( LE = () ) entonces

*Escribir(“no hay solución”), PARE;*

3. (F,P):= *Primer*(LE)

4. Si ( P es Meta ) entonces

*Determinar\_Solucion, PARE;*

#### Genera Sucesores:

5. *Adiciona\_ultimo*((F,P), LV);

6. *Elimina\_primer*(LE);

7. Hijos\_diferentes := Hijos(P) – LV

8. *Ordenar*(Hijos\_diferentes)

9. Para (Nodo  $\in$  (Hijos(P) - LV))

*Adicionar\_primero*((P,Nodo), LE);

10. Ir a 2

## 4.5.1 Método Subiendo a la Colina

---

### Observaciones

El método no garantiza una solución exacta para problemas de optimización

Esto es debido por que la estrategia de selección solo considera el mejor nodo sucesor y no el mejor de los nodos de la lista de espera (LE)

## 4.5.2 Método Primero el Mejor

---

### Procedimiento

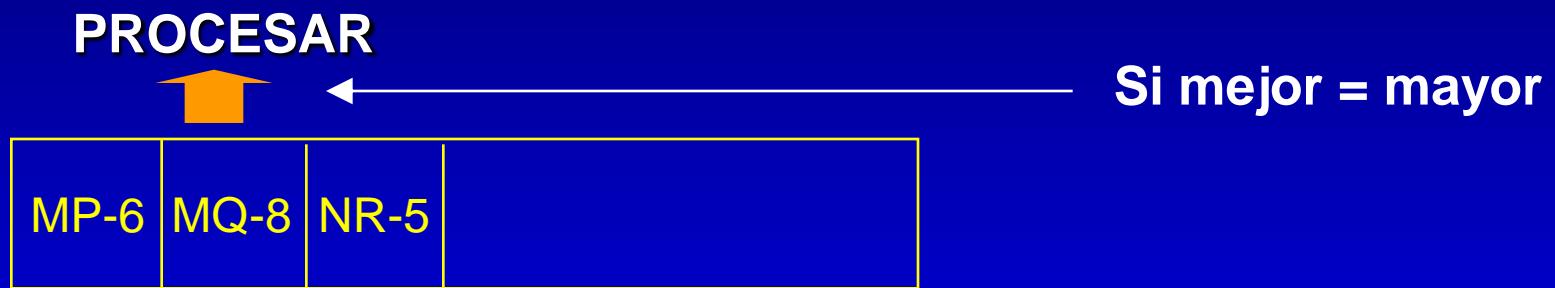
En este método el criterio de selección es dado por el nodo en LE que presenta “mejor” (mayor o menor) valor de la función de mérito.

Los nodos sucesores serán registrados como en profundidad, al inicio de LE, pero también pueden ser registrado al final, pues, el criterio de selección no depende del orden de como se registran los sucesores.

## 4.5.2 Método Primero el Mejor

### Implementación – Listas

LE:



LE:



## 4.5.2 Método Primero el Mejor

### Algoritmo - Listas

#### Inicio

1. LE := ((O,Estado\_Inicial)); LV:=();

#### Test de Parada

2. Si ( LE = () ) entonces

*Escribir(“no hay solución”), PARE;*

3. (F,P):= Mejor(LE)

4. Si ( P es Meta ) entonces

*Determinar\_Solucion, PARE;*

#### Genera Sucesores:

5. Adiciona\_ultimo((F,P), LV);

6. Elimina\_mejor(LE);

7. Para (Nodo ∈ (Hijos(P) - LV))

*Adicionar\_primer((P,Nodo), LE);*

8. Ir a 2

## 4.5.2 Método Primero el Mejor

---

### Observación

El método es adecuado para resolver problemas de optimización, garantiza solución óptima, pero puede requerir mayor número de operaciones.

## 4.5.3 Algoritmo A\*

---

### Procedimiento

Este algoritmo es similar al algoritmo “primero el mejor” con la única diferencia que la función evaluadora es definido como sigue:

$$f(e) = g(e) + h(e)$$

Donde:

$g(e)$  es la menor distancia (costo) desde el “estado inicial” hasta “e”

$h(e)$  es una sobre-estimación de la menor distancia (costo) de “e” al estado meta tal que  $h(\text{estado meta})=0$ .

## 4.5.3 Algoritmo A\*

---

### Estimación de $h$

#### Problema Puzzle

$h_1(e)$ : número de fichas que en el estado “e” se encuentran desordenados

$h_2(e)$ : suma de las distancias ortogonales de cada ficha respecto a su posición ordenada.

## 4.5.3 Algoritmo A\*

---



### Estimación de $h$

#### Problema Agente Viajero

$h_1(e)$ : número de ciudades por recorrer x distancia de la mayor arista no recorrida

$h_2(e)$ : número de ciudades por recorrer x la distancia promedio de las aristas no recorridas.

## 4.5.3 Algoritmo A\*

---

### Observación

**El algoritmo A\* es adecuado para resolver problemas de optimización, pero no garantiza solución óptima.**

---



## 4.6 Método de Ramificación y Poda

## 4.6 Método de Ramificación y Acotación

---

### Tópicos

- Conceptos: Ramificar y Acotar
- El Procedimiento de Ramificación y Acotación
- Resolución de Problemas

## 4.6.1. Conceptos: Ramificar y Acotar

---

### Ramificar

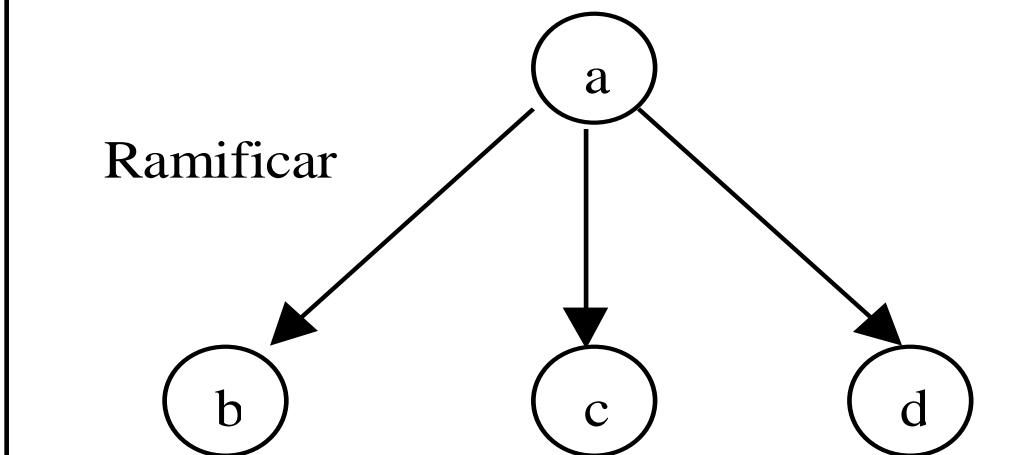
Se entiende por **ramificar** un nodo de un árbol al proceso de generar los nodos sucesores a este.

Todos los métodos ciegos y aquellos que usan información adicional son métodos de ramificación, pues todos ellos incluyen un proceso de ramificación

## 4.6.1 Conceptos: Ramificar y Acotar



### Ejemplo - Ramificar



Ramificación del nodo a

## 4.6.1. Conceptos: Ramificar y Acotar

---

### Criterios de Selección del Nodo a Ramificar

#### Primero el Mejor

El nodo que presenta “mejor” (mayor o menor) valor de la función de mérito será el primero a ser procesado

#### FIFO (First Input First Output)

El primer nodo a entrar en LE será el primero a ser procesado

#### LIFO (Last Input First Output)

El último nodo a entrar en LE será el primero a ser procesado

**Conveniencia: los nodos se registran al inicio**

## 4.6.1. Conceptos: Ramificar y Acotar



### Ejemplo - Criterio de Ramificación

LE

4	5	2	9	9	8
BC	BD	EF	AG	AC	FG

Nodo a Ramificar:

Primero el Mejor: AG o AC (mejor = mayor)

EF (mejor = menor)

LIFO: BC

FIFO: FG

## 4.6.1. Conceptos: Ramificar y Acotar

---

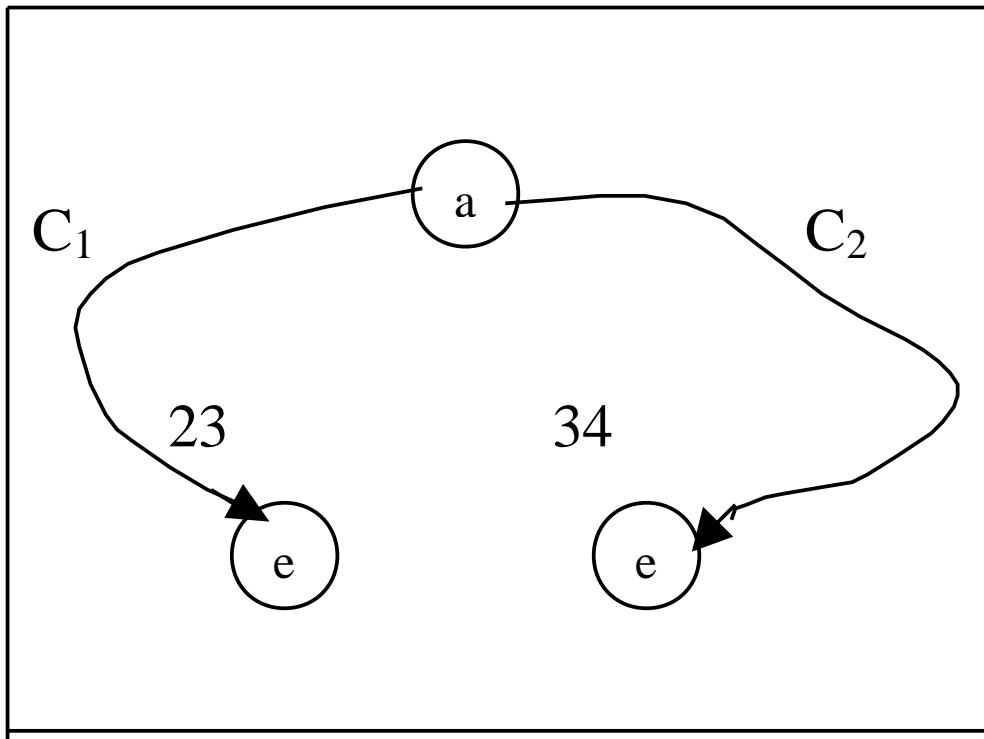
### Acotación (poda)

El proceso de simplificación de la búsqueda a través de la poda de ramas o sub-árboles del árbol de estado que presentan peores soluciones, es llamado de proceso de **acotación**.

## 4.6.1. Conceptos: Ramificar y Acotar



### Ejemplo de Acotación



Dos caminos conducen a un mismo nodo

## 4.6.1. Conceptos: Ramificar y Acotar

---

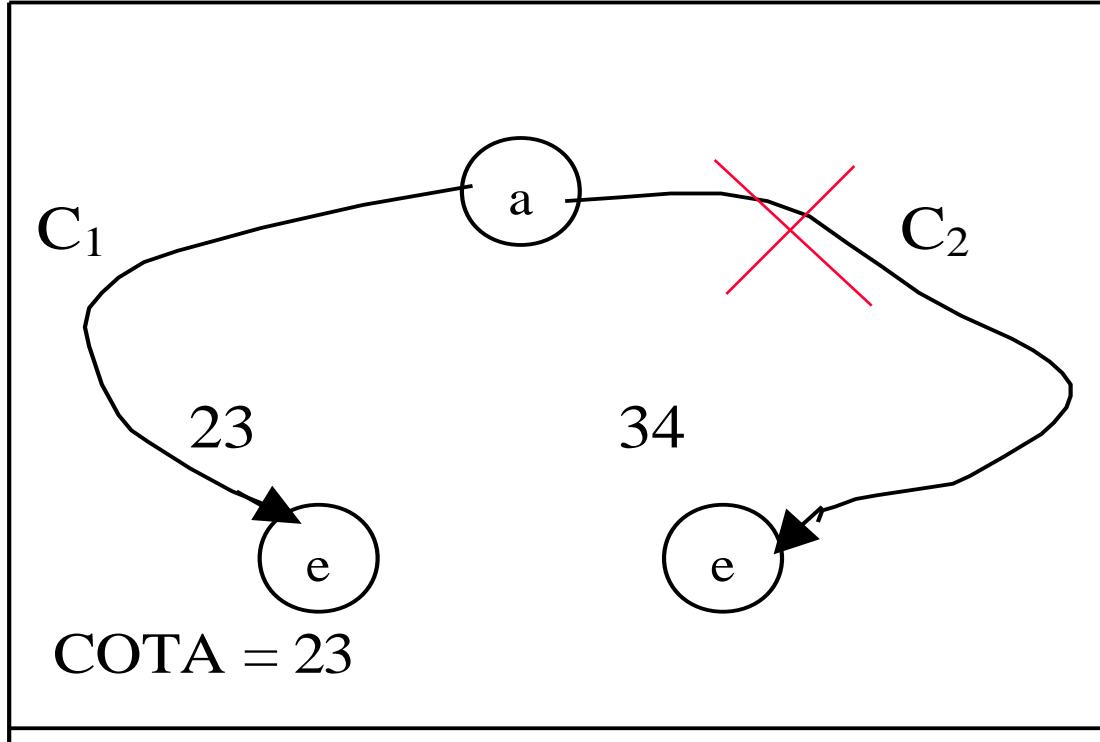
### Actualización de la Cota

Asociado a cada estado procesado se tiene un valor de la función de mérito, a este número se le llama de cota del nodo.

Si en el proceso de ramificación es generado un nodo ya procesado y con mejor valor de función de mérito que la cota asociada a este, entonces la cota de este nodo deberá ser actualizado por el valor de la función de mérito. En caso contrario, el nuevo nodo generado será excluido, porque generará igual o peor solución al existente.

## 4.6.1. Conceptos: Ramificar y Acotar

### Ejemplo de Acotación (problema de mínimo)

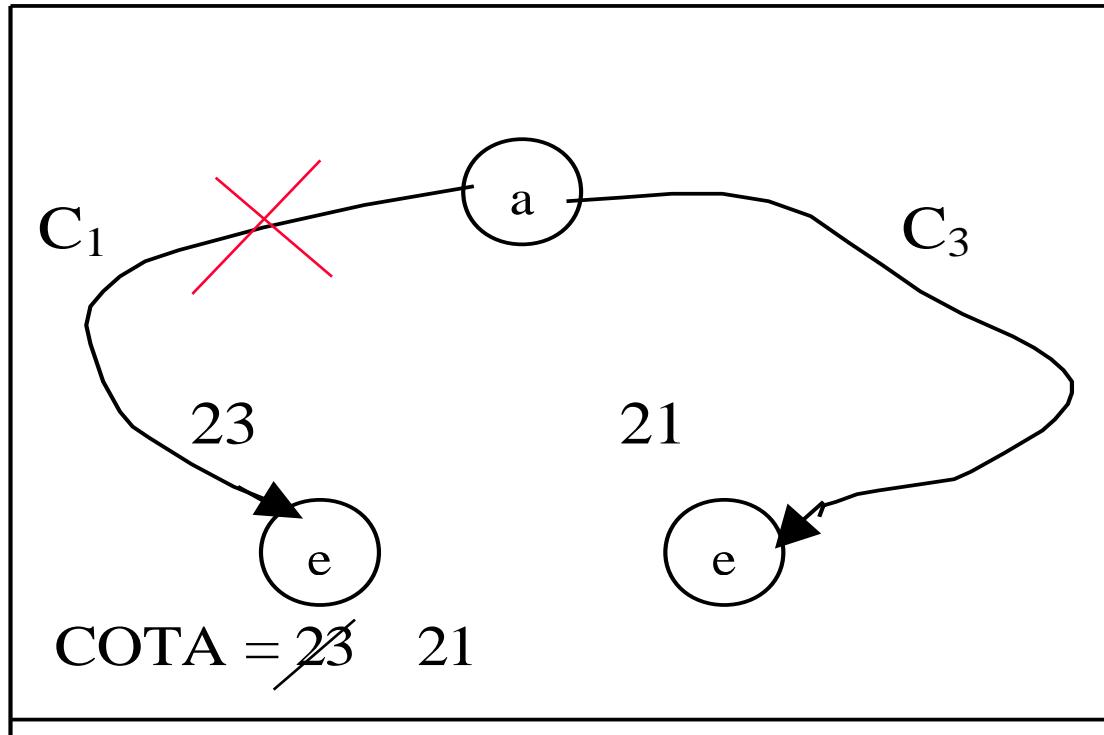


Dos caminos conducen a un mismo nodo

## 4.6.1. Conceptos: Ramificar y Acotar



### Ejemplo de Acotación (problema de mínimo)



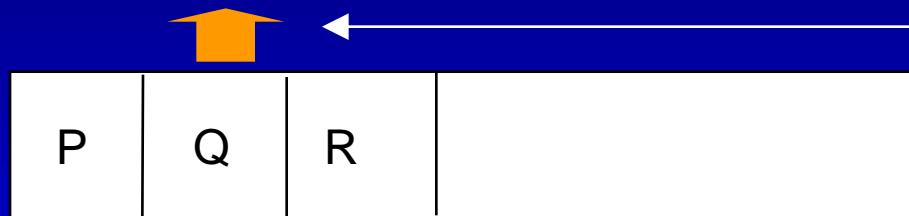
Dos caminos conducen a un mismo nodo

## 4.6.2. El Procedimiento de Ramificación y Acotación

### Implementación – Listas

LE:

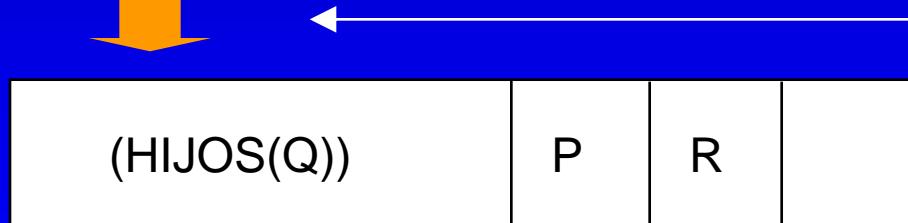
**PROCESAR**



Criterio de  
Ramificación

LE:

**REGISTRO**



Conveniencia

**ACTUALIZAR COTAS: LV**

## 4.6.2. El Procedimiento de Ramificación y Acotación

### Algoritmo de Ramificación y Acotación – Primero el Mejor

#### Inicio

1. LE := ((Estado\_Inicial)); LV := (Estado\_Inicial); Cota(Estado\_Inicial) := 0;

#### Test de Parada:

2. Si ( LE = () ) entonces Escribir("no hay solución"), PARE;

3. (F,P) := Mejor(LE); /\* F es el antecesor de P

4. Si ( P es Meta ) entonces Determina\_solucion, PARE;

#### Ramifica y Acota:

5. Elimina\_Mejor(LE);

6. Para ( Nodo  $\in$  Hijos(P) )

6.1 Si ( Nodo  $\notin$  LV)

6.2 Adiciona\_Inicio((P,Nodo)), LE)

6.3 Cota(Nodo) := f(Nodo)

6.4 Adiciona\_Ultimo((P,Nodo),LV)

Si\_No

6.5 Si ( f(Nodo) < Cota(Nodo) )

6.6 Adiciona\_Inicio(P,Nodo), LE)

6.7 Cota(Nodo) = f(Nodo)

6.8 Adiciona\_Ultimo((P,Nodo),LV)

Fin\_Si;

Fin\_Para;

7. Ir a 2

## 4.6.2. El Procedimiento de Ramificación y Acotación

---

### Observaciones

El método garantiza una solución exacta para problemas de optimización

Este es el método más eficiente de los métodos ciegos y de aquellos que usan información adicional.

## 4.6.3. Resolución de Problemas

---



Algoritmo de Propósito  
General

Algoritmo de Propósito  
Especial

## 4.6.3. Resolución de Problemas

---

### Algoritmo de Propósito General

Se refiere a un algoritmo que pueda servir para resolver cualquier problema de IA.

Para problemas de Optimización, el paso 4 (test de parada) deberá ser sustituido por alguna condición de optimalidad (condición que cuando verificada indica que una solución fue encontrada).

Este algoritmo en principio podrá resolver cualquiera problema de optimización, entretanto será por lo general altamente ineficiente

## 4.6.3. Resolución de Problemas

---

### Algoritmo de Propósito General

#### Parámetros de Entrada:

- Estado inicial
- Condición de optimalidad o Test de Parada
- Sistema de producción
- Función de evaluadora

## 4.6.3. Resolución de Problemas

---

### Algoritmo de Propósito Especial

Se refiere a un algoritmo desarrollado para resolver un problema específico de IA, no sirviendo para resolver algún otro problema.

Para problemas de optimización el test de parada (paso 4) deberá ser sustituido por alguna condición de optimalidad, y todos los otros pasos deberán ser modificados aprovechando la estructura particular y específica del problema a resolver.

Este algoritmo en general presenta buena eficiencia respecto a los algoritmos de propósito general, razón por la cual son los más usados para resolver problemas del tipo de optimización.

## 6.4.3. Resolución de Problemas

---

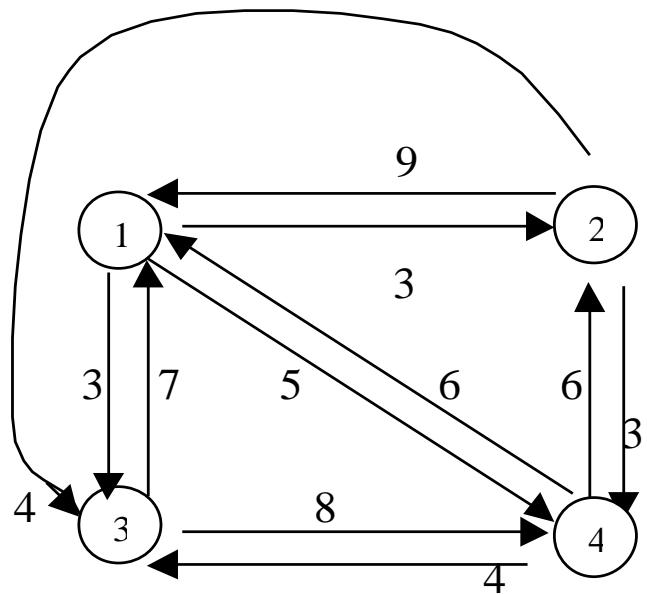
### Algoritmo de Propósito Especial

#### Parámetros de Entrada:

- Estado inicial
- Condición de optimalidad o Test de Parada
- Sistema de producción
- Función de evaluadora

## 6.4.3. Resolución de Problemas

### Agente Viajero - Representación



Ciudad de Origen	Ciudad de Destino			
	1	2	3	4
1	-	3	3	5
2	9	-	4	3
3	7	-	-	8
4	6	6	4	-

Red de transporte entre ciudades

## 6.4.3. Resolución de Problemas

---

### Agente Viajero

**Estado:**

Lista de ciudades que se visita

**Estado Inicial:**

Ciudad de partida

[1]

## 6.4.3. Resolución de Problemas

### Algoritmo de Propósito Especial Agente Viajero

#### Inicio

1. LE:=[1]; UB:= $+\infty$

#### Test de Parada:

2. Si ( LE = () ) entonces Escribir("no hay solución"), PARE;
3. LISTA := Mejor(LE);
4. P:= Ultimo(LISTA);
5. Si (L(LISTA) = n+1) entonces Escribir("solución=",LISTA), PARE;

#### Ramifica y Acota:

6. Elimina\_mejor(LE);
7. Para ( Nodo  $\in$  Hijos(P) )
  - 7.1      W\_L := LISTA;
  - 7.2      Adiciona\_ultimo(Nodo, W\_L);
  - 7.3      Si ( $f(W_L) < UB$ ) entonces
  - 7.4              Adiciona\_inicio(W\_L, LE)
  - 7.5              Si (L(W\_L)=n+1) entonces       $UB = f(W_L)$ ;
- Fin-Para
8. Ir Para 2