Inteligencia Artificial

Grupo PLG

Universidad Carlos III de Madrid

IA. 2008

Indice

Indice

Indice

Indice

Indice

Introducción

- Espacio de problemas
  - Conjunto de estados
  - Conjunto de operadores
  - Estado(s) inicial(es)
  - Meta(s)
- Representable por un grafo
- Resolución de problemas = búsqueda en el grafo
- Normalmente, la búsqueda genera un árbol
- Parámetros importantes
  - Factor de ramificación
  - Profundidad del árbol de búsqueda
- Diferencia con OR: el grafo en OR es explícito. En muchos problemas es imposible

Introducción

- Espacio de problemas
  - Conjunto de estados
  - Conjunto de operadores
  - Estado(s) inicial(es)
  - Meta(s)
- Representable por un grafo
- Resolución de problemas = búsqueda en el grafo
- Normalmente, la búsqueda genera un árbol
- Parámetros importantes
  - Factor de ramificación
  - Profundidad del árbol de búsqueda
- Diferencia con OR: el grafo en OR es explícito. En muchos problemas es imposible

Introducción

- Espacio de problemas
  - Conjunto de estados
  - Conjunto de operadores
  - Estado(s) inicial(es)
  - Meta(s)
- Representable por un grafo
- Resolución de problemas = búsqueda en el grafo
- Normalmente, la búsqueda genera un árbol
- Parámetros importantes
  - Factor de ramificación
  - Profundidad del árbol de búsqueda
- Diferencia con OR: el grafo en OR es explícito. En muchos problemas es imposible

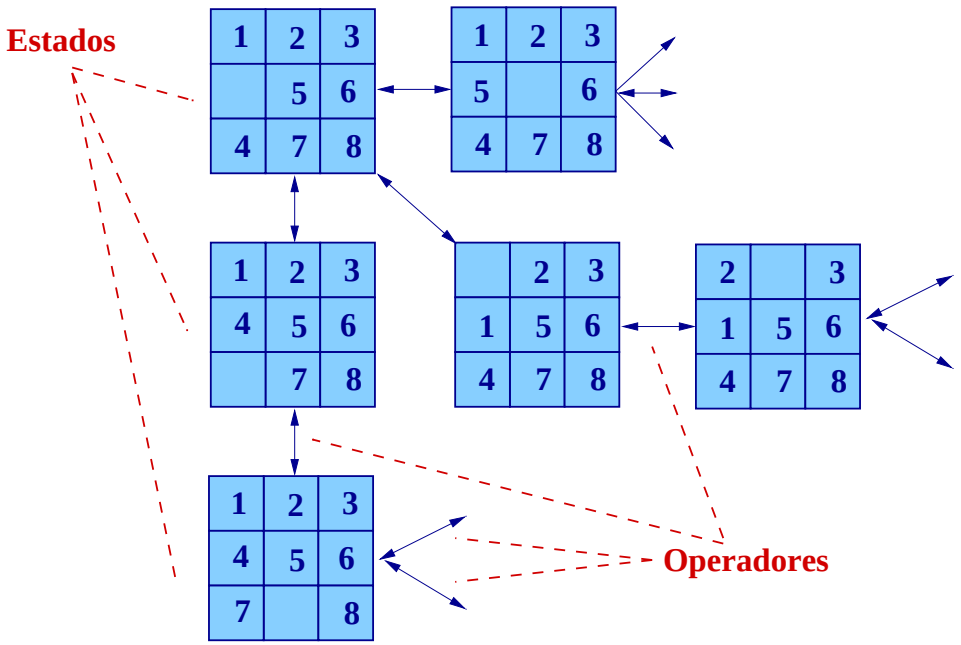
Introducción

- Espacio de problemas
  - Conjunto de estados
  - Conjunto de operadores
  - Estado(s) inicial(es)
  - Meta(s)
- Representable por un grafo
- Resolución de problemas = búsqueda en el grafo
- Normalmente, la búsqueda genera un árbol
- Parámetros importantes
  - Factor de ramificación
  - Profundidad del árbol de búsqueda
- Diferencia con OR: el grafo en OR es explícito. En muchos problemas es imposible

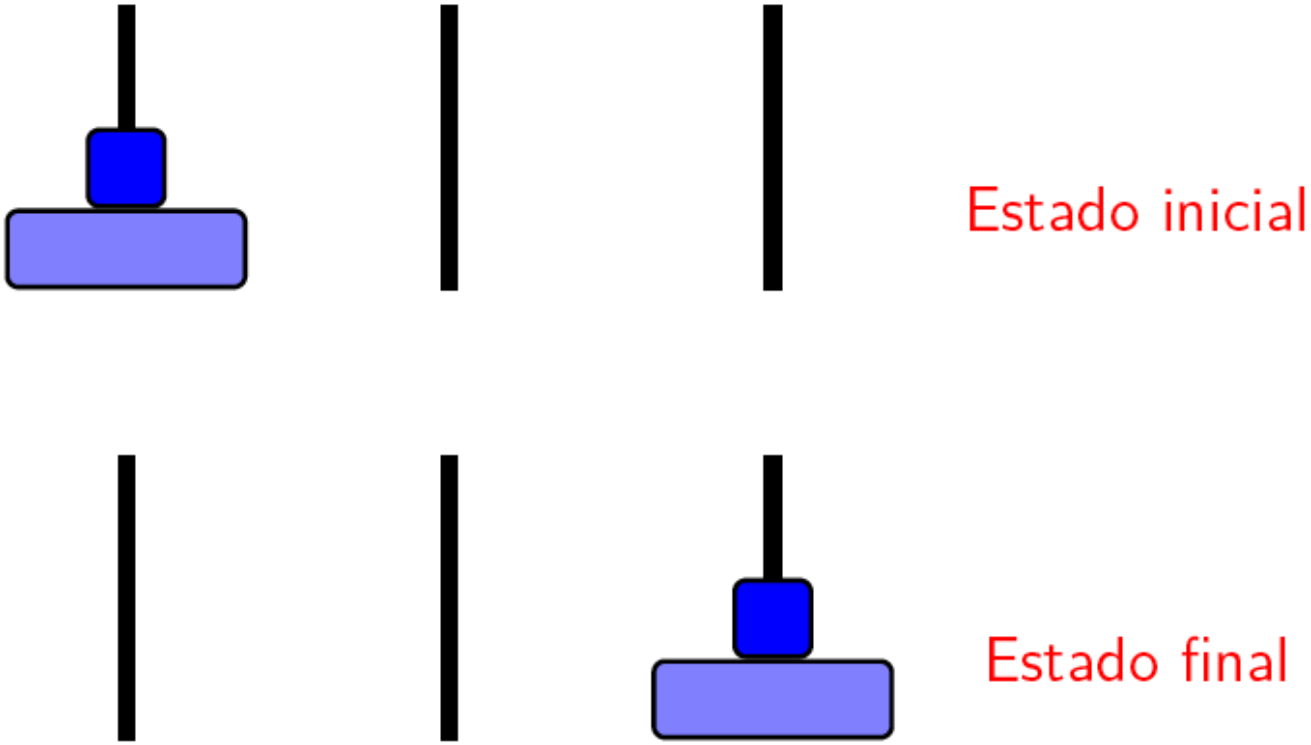
Introducción

- Espacio de problemas
  - Conjunto de estados
  - Conjunto de operadores
  - Estado(s) inicial(es)
  - Meta(s)
- Representable por un grafo
- Resolución de problemas = búsqueda en el grafo
- Normalmente, la búsqueda genera un árbol
- Parámetros importantes
  - Factor de ramificación
  - Profundidad del árbol de búsqueda
- Diferencia con OR: el grafo en OR es explícito. En muchos problemas es imposible

Ejemplo: 8-Puzzle



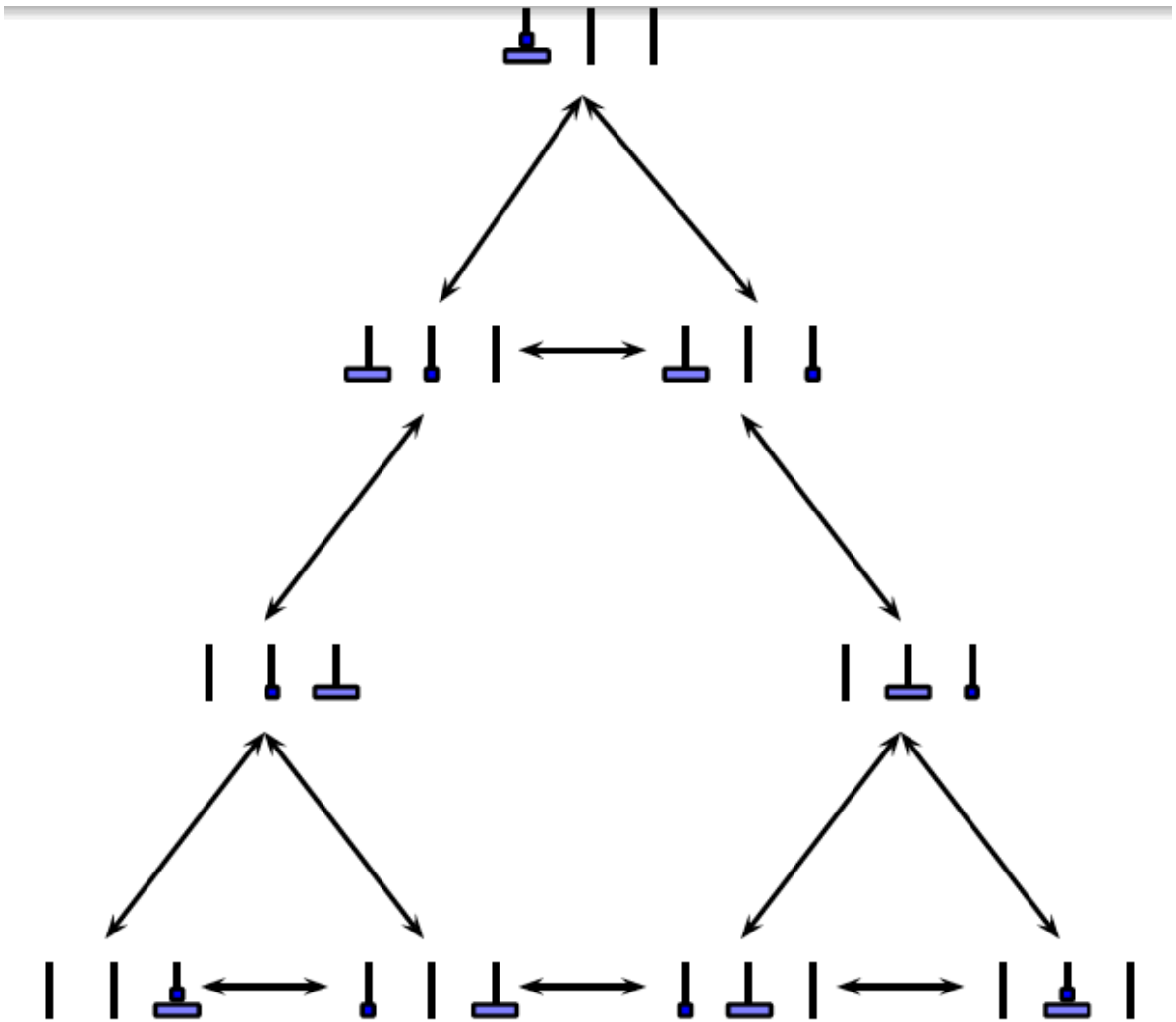
Ejemplo: Las torres de Hanoi (3,2)



Estado inicial

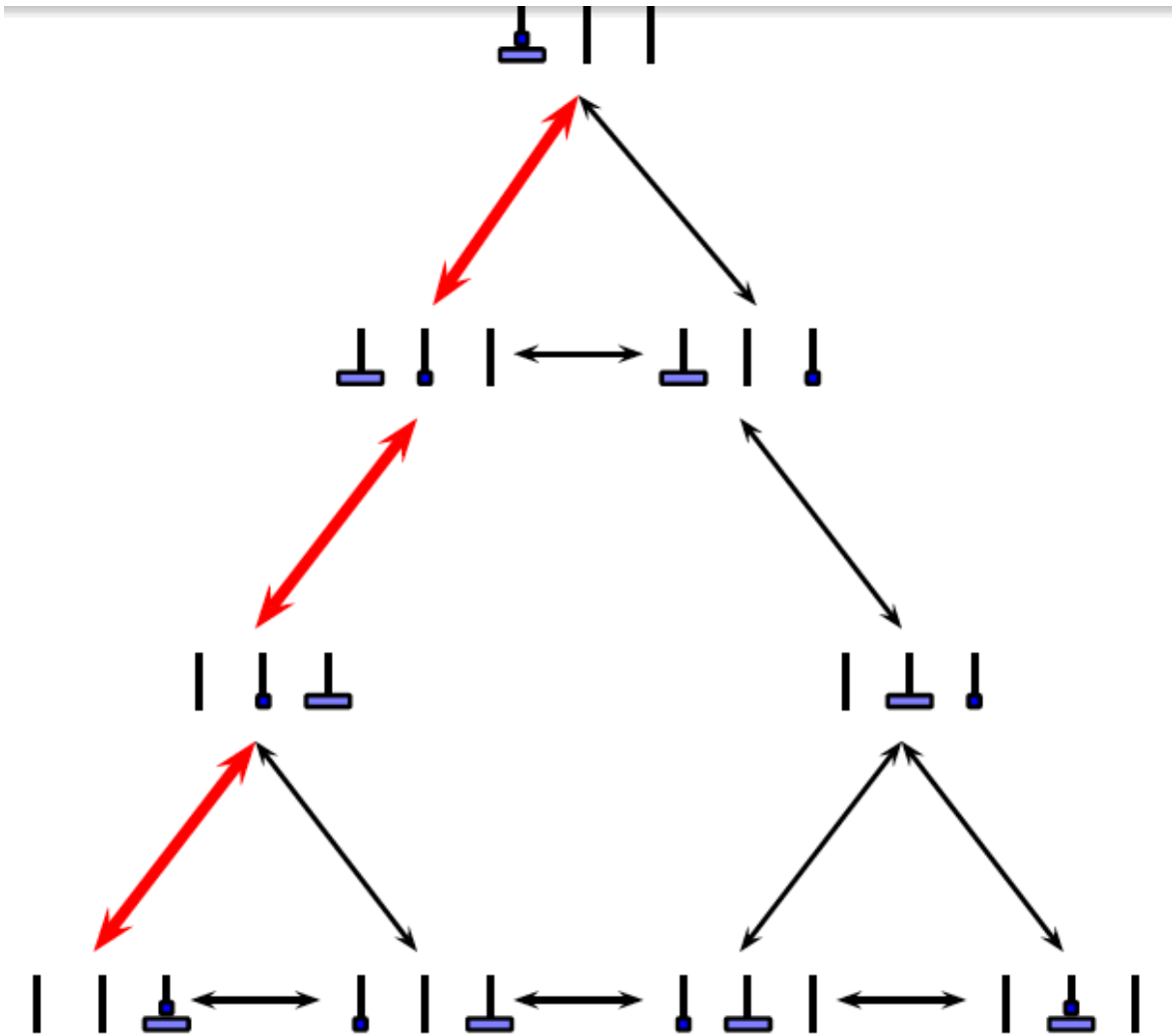
Estado final

Ejemplo: Las torres de Hanoi (3,2)



El *factor de ramificación* ($fr = \frac{8}{3}$) es una propiedad del grafo de estados

Ejemplo: Las torres de Hanoi (3,2)



La *profundidad* ($p = 3$) es una propiedad del problema a resolver

Ejemplo: Las jarras

Se tienen dos jarras de agua: una de cinco litros de capacidad y otra de tres. Ninguna de ellas tiene marcas de medición. Se tiene una bomba que permite llenar las jarras de agua, vaciarlas, y traspasar contenido de una jarra a otra.
¿Cómo se puede tener exactamente cuatro litros de agua en la jarra de cinco litros de capacidad?

Ejemplo: Las jarras

- Espacio de Estados:
  - conjunto de pares ordenados de enteros (x, y), de forma que $x = 0, \ldots, 5$, $y = 0, \ldots, 3$
  - x representa el número de litros de agua que hay en la jarra de 5 litros de capacidad
  - y representa el número de litros de agua que hay en la jarra de 3 litros de capacidad
- Estado inicial: $(0, 0)$
- Estado meta:
  - Descripción implícita: $(4, n)$, donde $n = 0, \ldots, 3$
  - Descripción explícita: $(4, 0)$, $(4, 1)$, $(4, 2)$, $(4, 3)$

Ejemplo: Las jarras

Operadores

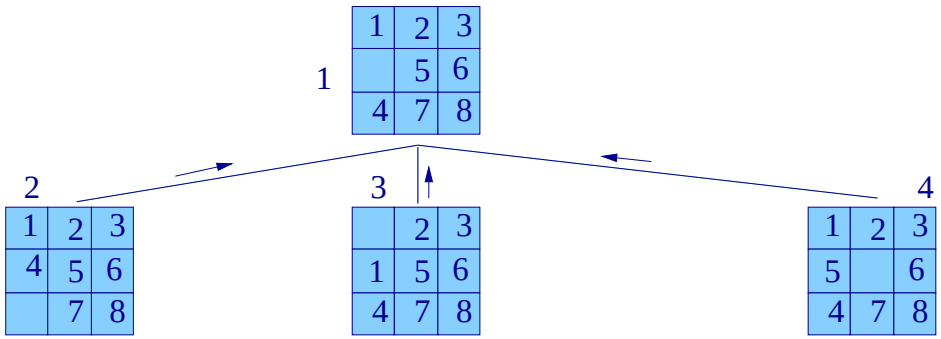
Llenar jarra grande : Si $(x < 5) \rightarrow (5, y)$
Llenar jarra pequeña : Si $(y < 3) \rightarrow (x, 3)$
Vaciar jarra grande : Si $(x > 0) \rightarrow (0, y)$
Vaciar jarra pequeña : Si $(y > 0) \rightarrow (x, 0)$
Verter en grande : Si $(y > 0) \rightarrow (x + \text{mín}\{5 - x, y\}, y - \text{mín}\{5 - x, y\})$
Verter en pequeña : Si $(x > 0) \rightarrow (x - \text{mín}\{x, 3 - y\}, y + \text{mín}\{x, 3 - y\})$

Explosión combinatoria

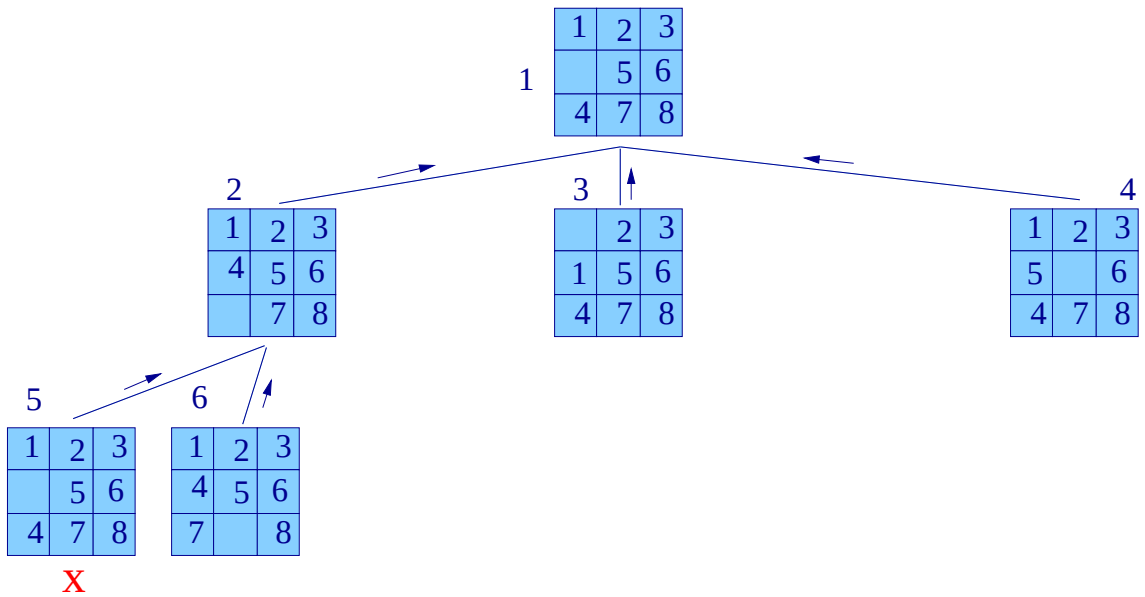
Dominio	**Número de estados**	**Tiempo (10^7 nodos/s)**
8-puzzle	$\left(\frac{N^2!}{2}\right)\Big\vert_{N=3} = 181{,}440$	0.01 segundos
15-puzzle	$\left(\frac{N^2!}{2}\right)\Big\vert_{N=4} = 10^{13}$	11,5 días
24-puzzle	$\left(\frac{N^2!}{2}\right)\Big\vert_{N=5} = 10^{25}$	$31,7 \times 10^9$ años
Hanoi (3,2)	$(3^n)\vert_{n=2} = 9$	9×10^{-7} segundos
Hanoi (3,4)	$(3^n)\vert_{n=4} = 81$	$8,1 \times 10^{-6}$ segundos
Hanoi (3,8)	$(3^n)\vert_{n=8} = 6561$	$6,5 \times 10^{-4}$ segundos
Hanoi (3,16)	$(3^n)\vert_{n=16} = 4,3 \times 10^7$	4,3 segundos
Hanoi (3,24)	$(3^n)\vert_{n=24} = 2,824 \times 10^11$	0,32 días
Cubo de Rubik $2 \times 2 \times 2$	10^6	0,1 segundos
Cubo de Rubik $3 \times 3 \times 3$	$4,32 \times 10^{19}$	31.000 años

Indice

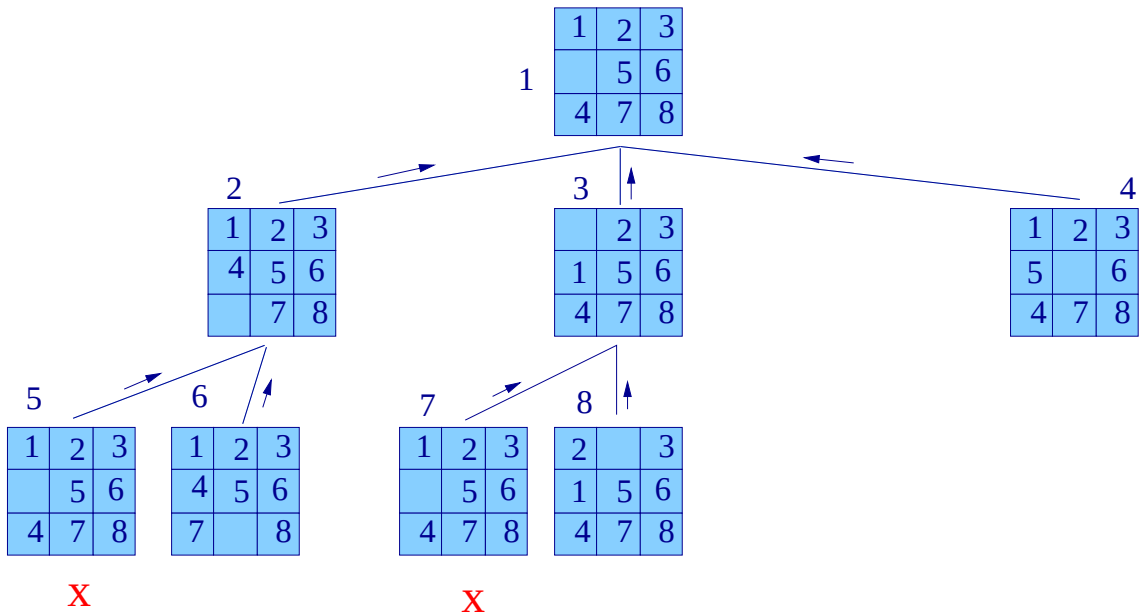
8-Puzzle – Amplitud



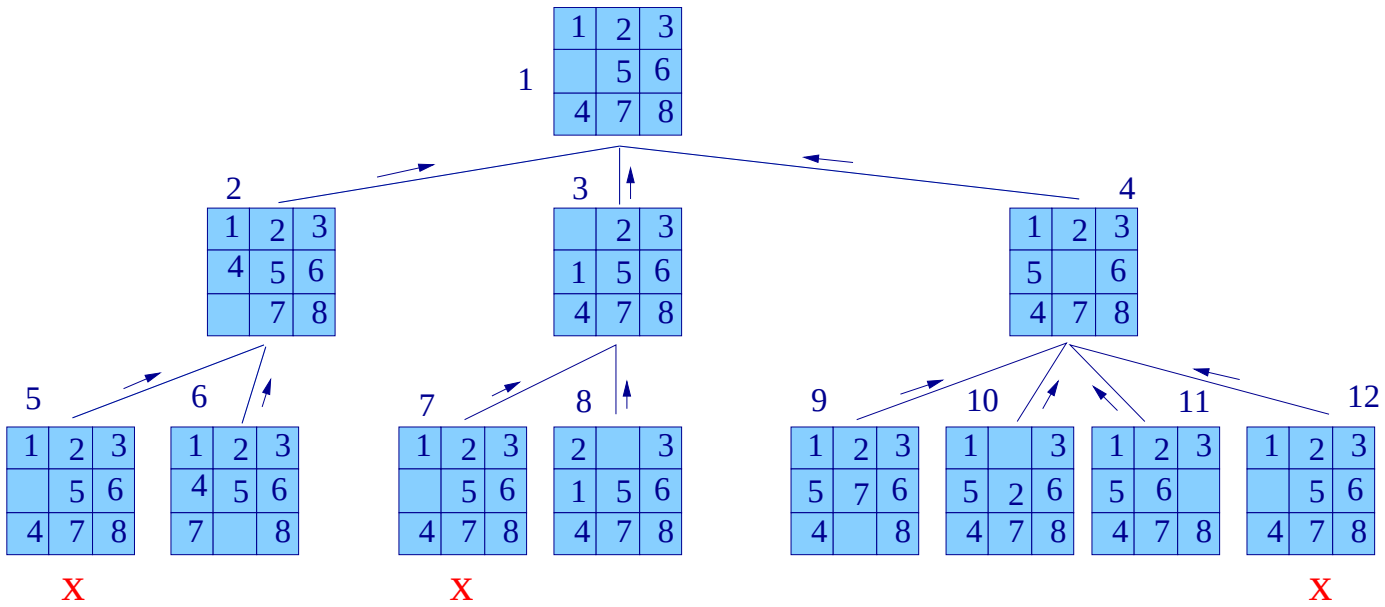
8-Puzzle – Amplitud



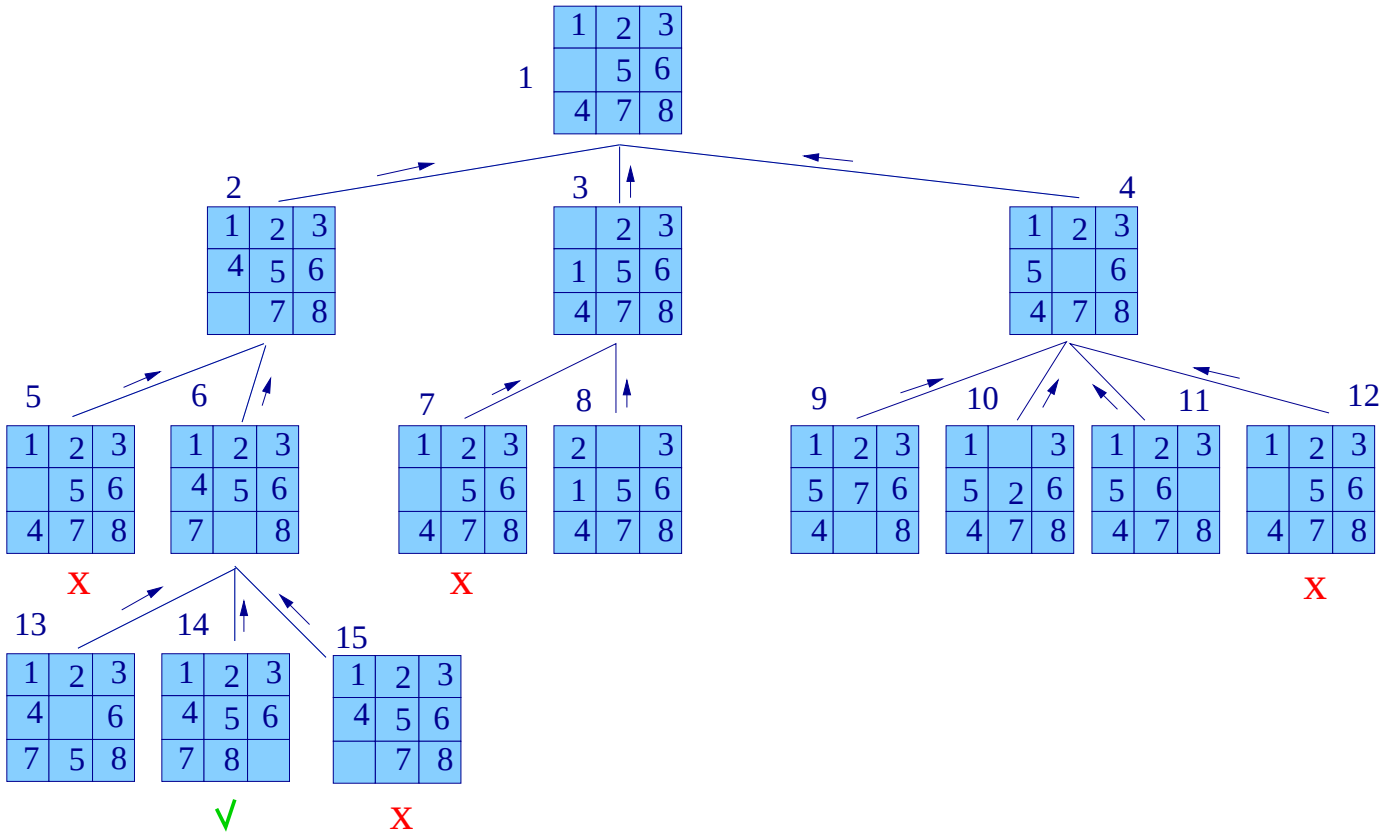
8-Puzzle – Amplitud



8-Puzzle – Amplitud



8-Puzzle – Amplitud



Búsqueda en amplitud

Procedimiento Amplitud (Estado-inicial Estado-Final)

1. Crear lista ABIERTA con el nodo inicial, I, (estado-inicial)
2. EXITO=Falso
3. Hasta que ABIERTA esté vacía O EXITO

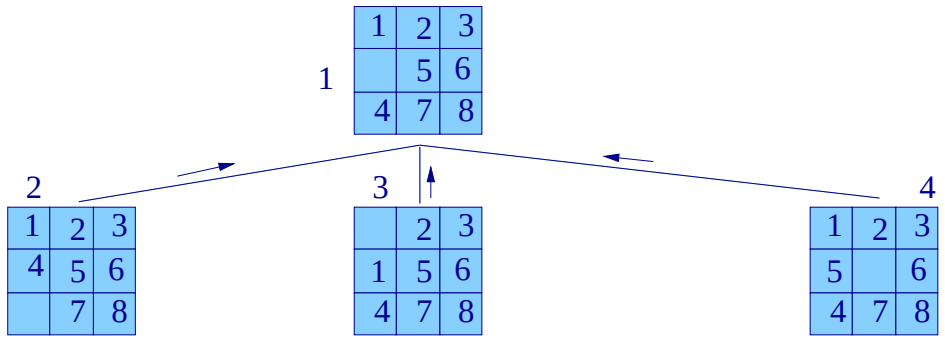
   Quitar de ABIERTA el primer nodo, N
   Si N tiene sucesores
   Entonces Generar los sucesores de N
   Crear punteros desde los sucesores hacia N
   Si algún sucesor es nodo meta
   Entonces EXITO=Verdadero
   Si no Añadir los sucesores al final de ABIERTA

4. Si EXITO
   Entonces Solución=camino desde I a N por los punteros
   Si no, Solución=fracaso

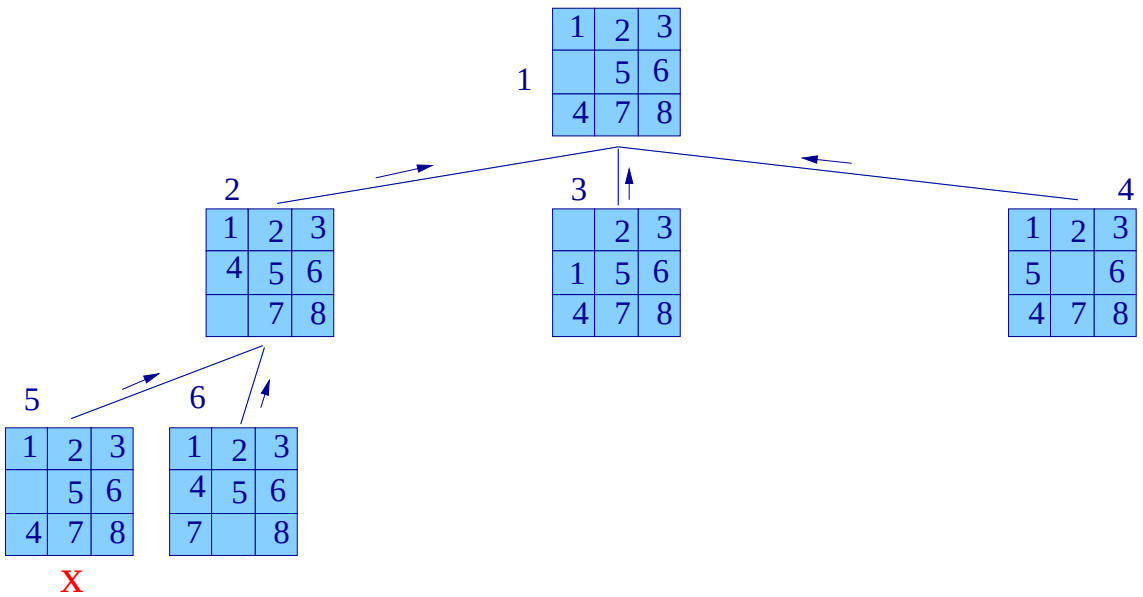
Características

- **Completo**: encuentra solución si existe y el factor de ramificación es finito en cada nodo
- **Optimalidad**: si todos los operadores tienen el mismo coste, encontrará la solución óptima
- **Eficiencia**: buena si las metas están cercanas
- Problema: consume memoria exponencial

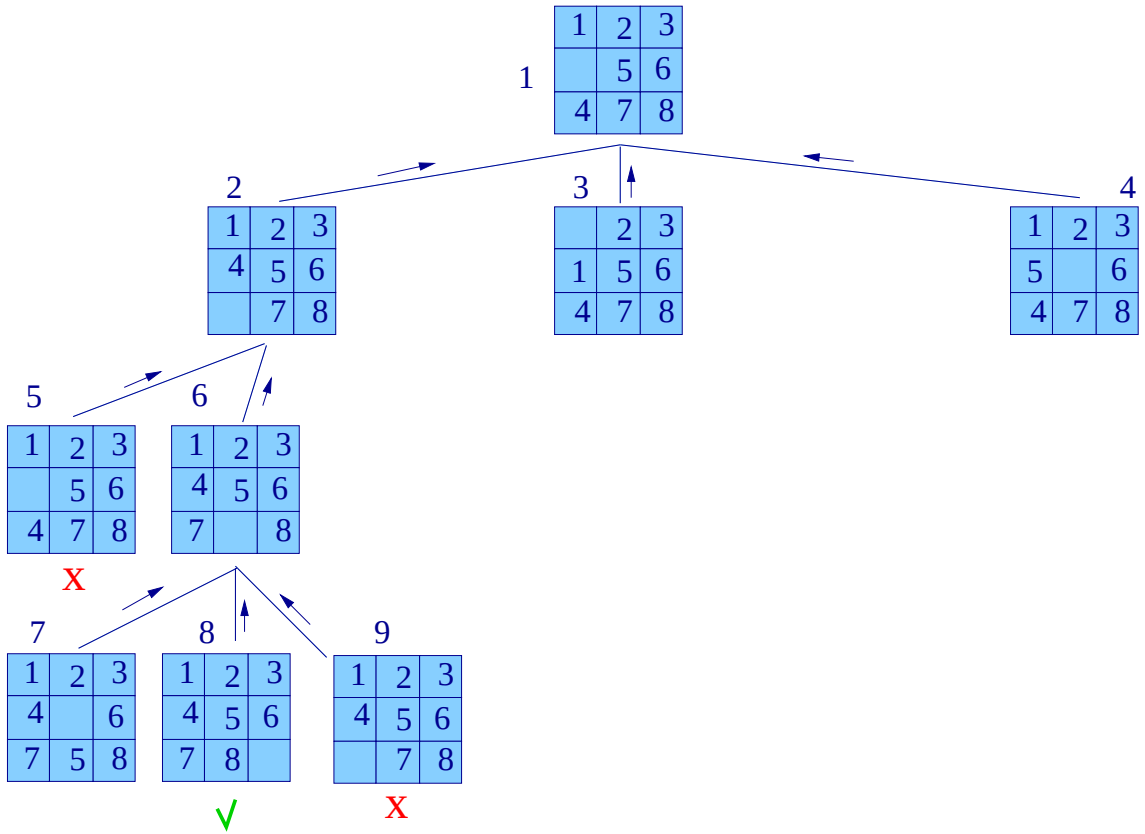
8-Puzzle – Profundidad



8-Puzzle – Profundidad



8-Puzzle – Profundidad



Búsqueda en profundidad

Procedimiento Profundidad (Estado-inicial,Estado-Final,Profundidad-máxima)

1. Crear lista ABIERTA con el nodo inicial, I, y su profundidad=0
2. EXITO=Falso
3. Hasta que ABIERTA esté vacía O EXITO

   Quitar de ABIERTA el primer nodo.
   Lo llamaremos N y a su profundidad P
   Si P $<$ Profundidad-máxima Y N tiene sucesores
   Entonces Generar los sucesores de N
   Crear punteros desde los sucesores hacia N
   Si algún sucesor es el Estado-Final
   Entonces EXITO=Verdadero
   Si no, Añadir los sucesores al principio de ABIERTA
   Asignarles profundidad P+1

4. Si EXITO
   Entonces Solución=camino desde I a N por los punteros
   Si no, Solución=fracaso

Características

- Requiere técnica de retroceso ("backtracking")
- Razones para retroceso:
  - Se ha llegado al límite de profundidad
  - Se han estudiado todos los sucesores de un nodo y no se ha llegado a la solución
  - Se sabe que el estado no conduce a la solución
  - Se genera un estado repetido
- **Completitud**: no asegura encontrar la solución
- **Optimalidad**: no asegura encontrar la solución óptima
- **Eficiencia**: bueno cuando metas alejadas de estado inicial, o problemas de memoria
- No es bueno cuando hay ciclos

Características

- Requiere técnica de retroceso ("backtracking")
- Razones para retroceso:
  - Se ha llegado al límite de profundidad
  - Se han estudiado todos los sucesores de un nodo y no se ha llegado a la solución
  - Se sabe que el estado no conduce a la solución
  - Se genera un estado repetido
- **Completitud**: no asegura encontrar la solución
- **Optimalidad**: no asegura encontrar la solución óptima
- **Eficiencia**: bueno cuando metas alejadas de estado inicial, o problemas de memoria
- No es bueno cuando hay ciclos

Características

- Requiere técnica de retroceso ("backtracking")
- Razones para retroceso:
  - Se ha llegado al límite de profundidad
  - Se han estudiado todos los sucesores de un nodo y no se ha llegado a la solución
  - Se sabe que el estado no conduce a la solución
  - Se genera un estado repetido
- **Completitud**: no asegura encontrar la solución
- **Optimalidad**: no asegura encontrar la solución óptima
- **Eficiencia**: bueno cuando metas alejadas de estado inicial, o problemas de memoria
- No es bueno cuando hay ciclos

Búsqueda no informada con costes

- Dijkstra: amplitud, pero eligiendo aquel nodo que tenga menor coste desde el nodo raíz
- *Branch and bound, B&B*
  - búsqueda (normalmente en profundidad)
  - cuando encuentra una solución, su coste se convierte en un límite (superior o inferior) para los siguientes nodos
  - si un nodo posterior tiene un coste desde el nodo raíz mayor/menor (o igual) que el límite, se termina la búsqueda por él
  - si se tiene un límite inicial, se puede utilizar la misma idea para amplitud/Dijkstra

Análisis de complejidad

- Si se dispone de:
  - factor de ramificación medio fr
  - profundidad del árbol de búsqueda p
- ¿Cuál sería, en el peor de los casos, el número de nodos que examinaría cada técnica?
  - Unidireccional
- Pista:
  - supóngase que fr=3, y vaya incrementándose la p
  - calcular de forma inductiva el número de nodos

Análisis de complejidad

Técnica de Búsqueda	Número máximo de nodos
Unidireccional	$\sum_{i=0}^{p} fr^i$

Análisis de complejidad

	Complejidad	
Técnica de Búsqueda	Temporal	Espacial
Amplitud	$O(fr^p)$	$O(fr^p)$
Profundidad	$O(fr^p)$	$O(p)$

- Cuando complejidad en tiempo es igual a la de espacio, se agota la memoria antes que el tiempo

Indice

Heurística

- Si se tiene conocimiento perfecto $\longrightarrow$ algoritmo exacto
- Si no se tiene conocimiento $\longrightarrow$ búsqueda sin información
- En la mayor parte de los problemas que resuelven los humanos, se está en posiciones intermedias
- Heurística: (del griego "*heurisko*" ($\varepsilon\acute{\upsilon}\rho\iota\sigma\kappa\omega$): "**yo encuentro**") conocimiento parcial sobre un problema/dominio que permite resolver problemas eficientemente en ese problema/dominio
- Representación de las heurísticas
  - funciones $h(n)$
  - metareglas
- Las funciones heurísticas se **descubren** resolviendo modelos *simplificados* del problema real

Heurística

- Si se tiene conocimiento perfecto $\longrightarrow$ algoritmo exacto
- Si no se tiene conocimiento $\longrightarrow$ búsqueda sin información
- En la mayor parte de los problemas que resuelven los humanos, se está en posiciones intermedias
- Heurística: (del griego "*heurisko*" ($\varepsilon\acute{\upsilon}\rho\iota\sigma\kappa\omega$): "**yo encuentro**") conocimiento parcial sobre un problema/dominio que permite resolver problemas eficientemente en ese problema/dominio
- Representación de las heurísticas
  - funciones $h(n)$
  - metareglas
- Las funciones heurísticas se **descubren** resolviendo modelos *simplificados* del problema real

Heurística

- Si se tiene conocimiento perfecto $\longrightarrow$ algoritmo exacto
- Si no se tiene conocimiento $\longrightarrow$ búsqueda sin información
- En la mayor parte de los problemas que resuelven los humanos, se está en posiciones intermedias
- Heurística: (del griego "*heurisko*" ($\varepsilon\acute{\upsilon}\rho\iota\sigma\kappa\omega$): "**yo encuentro**") conocimiento parcial sobre un problema/dominio que permite resolver problemas eficientemente en ese problema/dominio
- Representación de las heurísticas
  - funciones $h(n)$
  - metareglas
- Las funciones heurísticas se **descubren** resolviendo modelos *simplificados* del problema real

Heurística

- Si se tiene conocimiento perfecto $\longrightarrow$ algoritmo exacto
- Si no se tiene conocimiento $\longrightarrow$ búsqueda sin información
- En la mayor parte de los problemas que resuelven los humanos, se está en posiciones intermedias
- Heurística: (del griego "*heurisko*" ($\varepsilon\acute{\upsilon}\rho\iota\sigma\kappa\omega$): "**yo encuentro**") conocimiento parcial sobre un problema/dominio que permite resolver problemas eficientemente en ese problema/dominio
- Representación de las heurísticas
  - funciones $h(n)$
  - metareglas
- Las funciones heurísticas se **descubren** resolviendo modelos *simplificados* del problema real

Problemas relajados. 8-puzzle

- Restricciones:
  - sólo se puede mover el blanco
  - el movimiento sólo se puede hacer a casillas adyacentes horizontal o vertical
  - en cada paso, se intercambian los contenidos de dos casillas
- Relajaciones:
  - si quitamos las dos primeras restricciones, generamos la heurística de número de casillas mal colocadas
  - si quitamos la primera restricción, generamos la heurística de la distancia de Manhattan

Problemas relajados. 8-puzzle

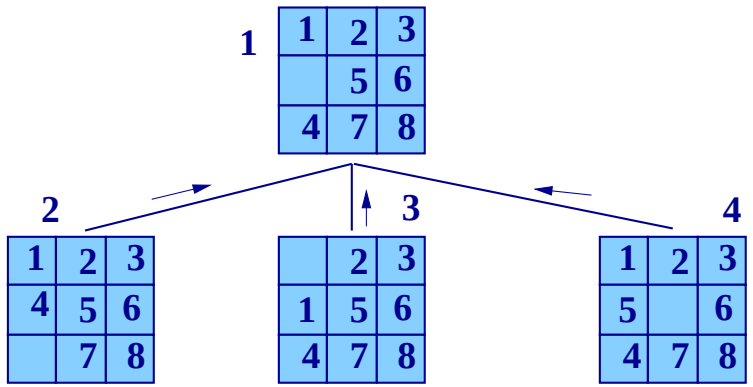
- Restricciones:
  - sólo se puede mover el blanco
  - el movimiento sólo se puede hacer a casillas adyacentes horizontal o vertical
  - en cada paso, se intercambian los contenidos de dos casillas
- Relajaciones:
  - si quitamos las dos primeras restricciones, generamos la heurística de número de casillas mal colocadas
  - si quitamos la primera restricción, generamos la heurística de la distancia de Manhattan

Algoritmo de escalada

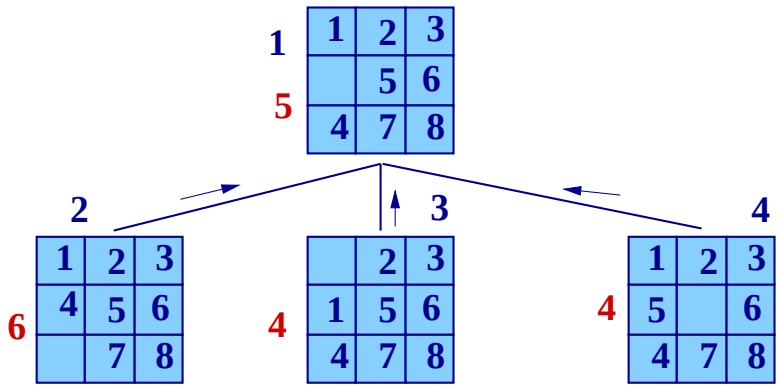
Procedimiento escalada (Estado-inicial Estado-final)

N=Estado-inicial; EXITO=Falso
Hasta que Camino-Sin-Salida(N) O EXITO
    Generar los sucesores de N
    SI algún sucesor es Estado-final
    ENTONCES EXITO=Verdadero
    SI NO, Evaluar cada nodo con la función de evaluación, $f(n)$
        N=mejor sucesor
Si EXITO
Entonces Solución=camino desde nodo del Estado-inicial
                  al nodo N por los punteros
Si no, Solución=fracaso

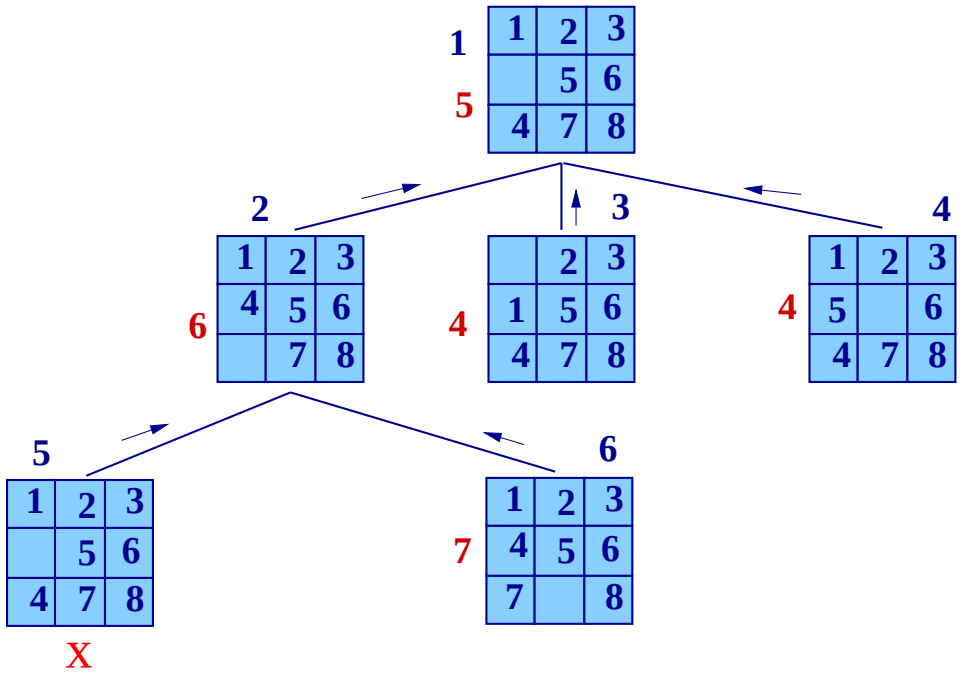
8-Puzzle – Búsqueda en escalada



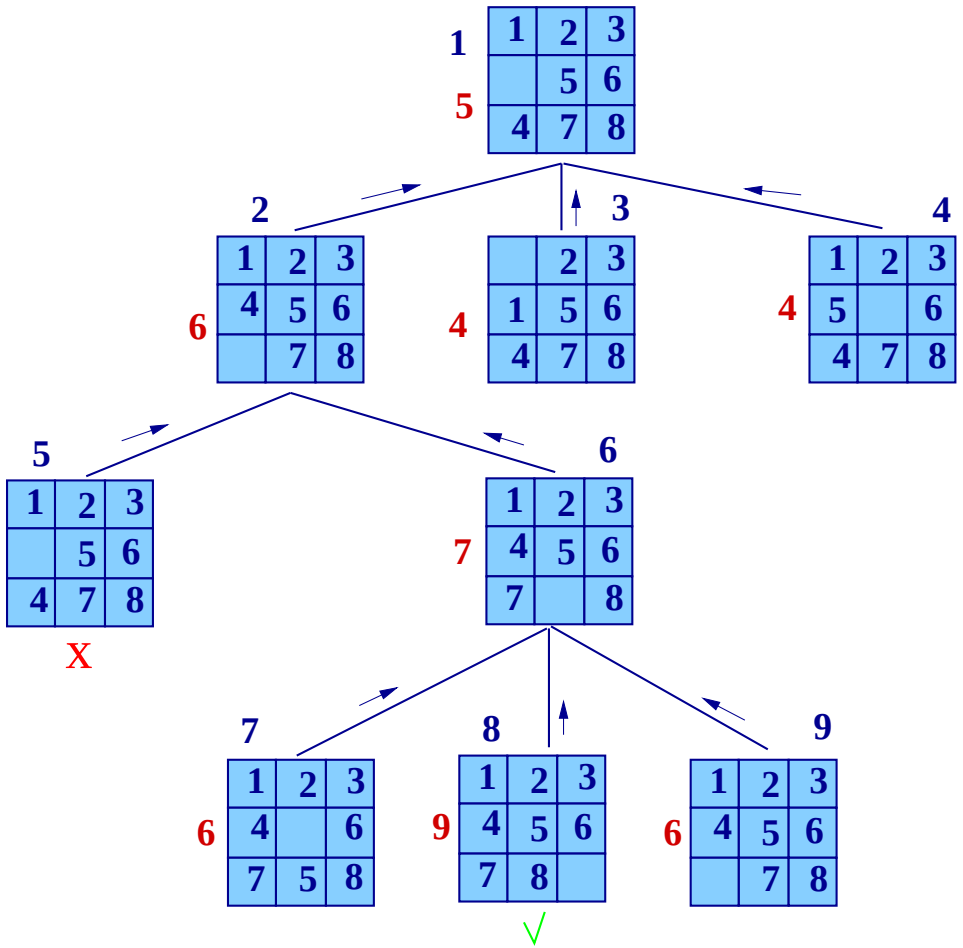
8-Puzzle – Búsqueda en escalada



8-Puzzle – Búsqueda en escalada

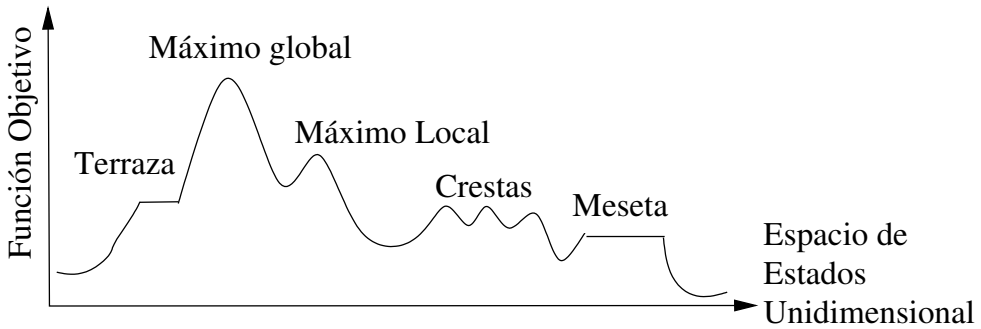


8-Puzzle – Búsqueda en escalada



Características

- Problemas de los métodos *avariciosos*
  - **Máximos (o mínimos) locales**: pico que es más alto que cada uno de sus estados vecinos, pero más bajo que el máximo global
  - **Mesetas**: zona del espacio de estados con función de evaluación plana
  - **Crestas**: zona del espacio de estados con varios máximos (mínimos) locales



Características

- Soluciones
  - Retroceso
  - Dar más de un paso
  - Reinicio aleatorio
- Método local
  - **Completitud**: no tiene porqué encontrar la solución
  - **Admisibilidad**: no siendo completo, aún menos será admisible
  - **Eficiencia**: rápido y útil si la función es monótona (de)creciente

Técnicas de mejor primero

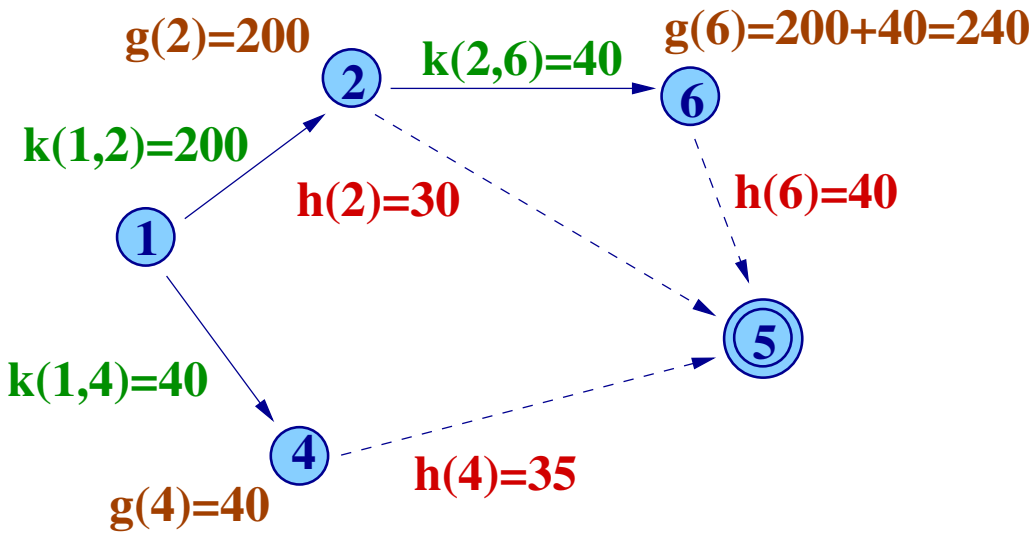
Procedimiento Mejor-primero (Estado-inicial Estado-final)

Crear grafo de búsqueda G, con el nodo inicial, I (Estado-inicial)
ABIERTA=I, CERRADA=Vacío, EXITO=Falso
Hasta que ABIERTA esté vacía O EXITO
    Quitar el primer nodo de ABIERTA, N y meterlo en CERRADA
    SI N es Estado-final ENTONCES EXITO=Verdadero
    SI NO Expandir N, generando el conjunto S de sucesores de N,
                que no son antecesores de N en el grafo
        Generar un nodo en G por cada s de S
        Establecer un puntero a N desde aquellos s de S que no estuvieran ya en G
        Añadirlos a ABIERTA
        Para cada s de S que estuviera ya en ABIERTA o CERRADA
                decidir si redirigir o no sus punteros hacia N
        Para cada s de S que estuviera ya en CERRADA
                decidir si redirigir o no los punteros de los nodos en sus subárboles
        Reordenar ABIERTA según $f(n)$
Si EXITO Entonces Solución=camino desde I a N a través de los punteros de G
Si no Solución=Fracaso

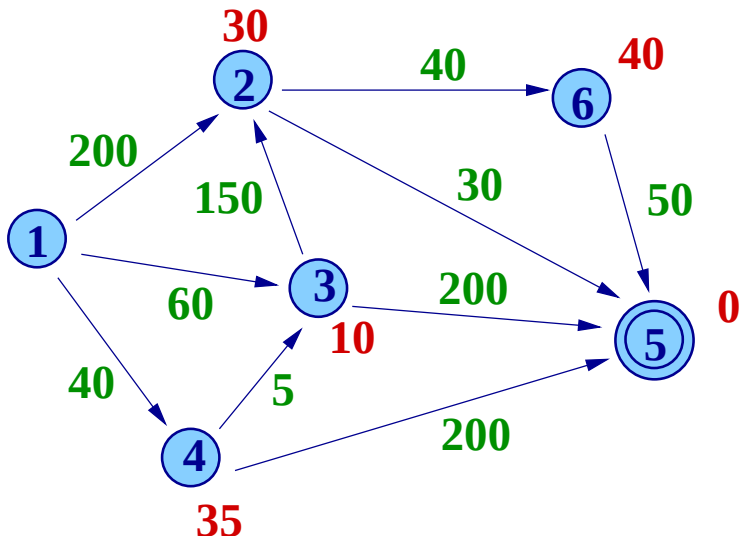
A* (Hart, Nilsson y Rafael, 1968)

- Función de ordenación de nodos: $f(n) = g(n) + h(n)$
  - $f(n)$: función de evaluación
  - $g(n)$: función de coste de ir desde el nodo inicial al nodo n
  - $h(n)$: función heurística que mide la distancia estimada desde n a algún nodo meta
- $g(n)$ se calcula como la suma de los costes de los arcos recorridos, $k(n_i, n_j)$
- Los valores reales sólo se pueden conocer al final de la búsqueda
  - $f^*(n)$: coste real para ir desde el nodo inicial a algún nodo meta a través de n
  - $g^*(n)$: coste real para ir desde el nodo inicial al nodo n
  - $h^*(n)$: coste real para ir desde el nodo n a algún nodo meta

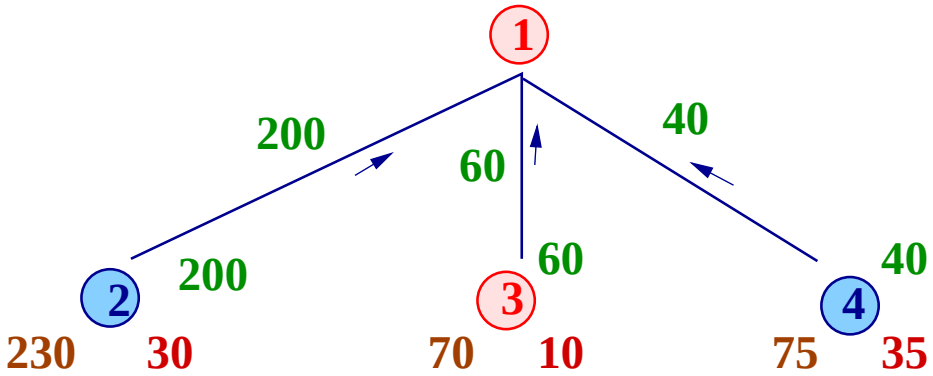
Definición de términos



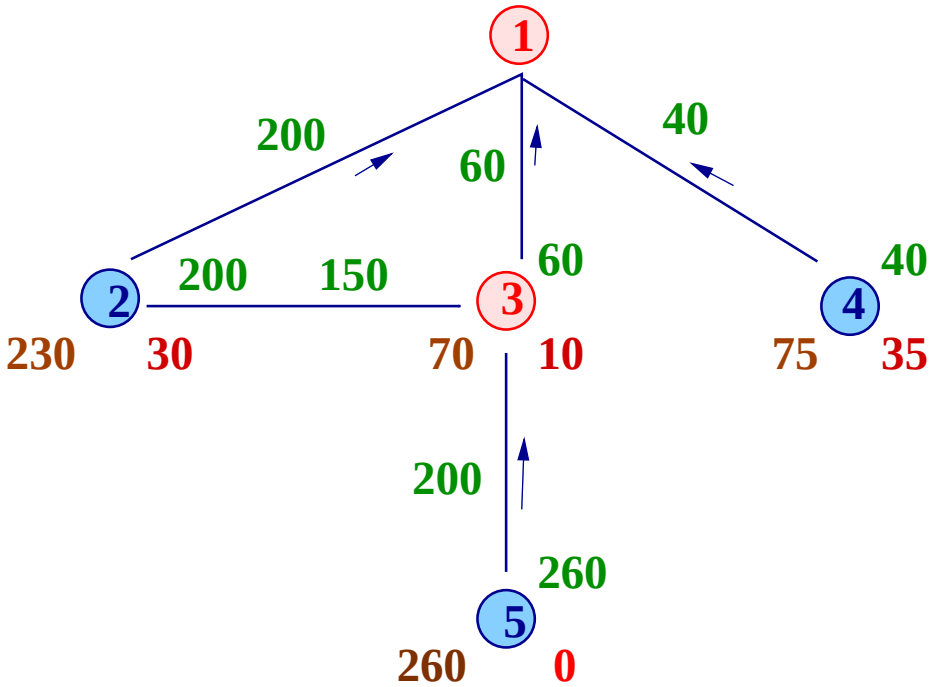
Ejemplo A*



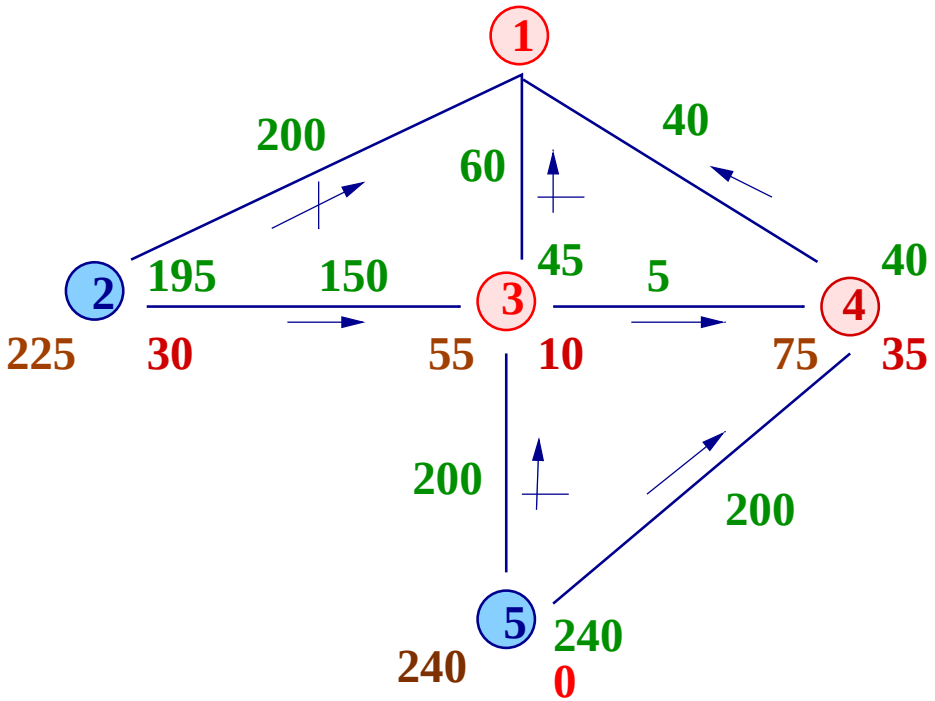
Ejemplo A*



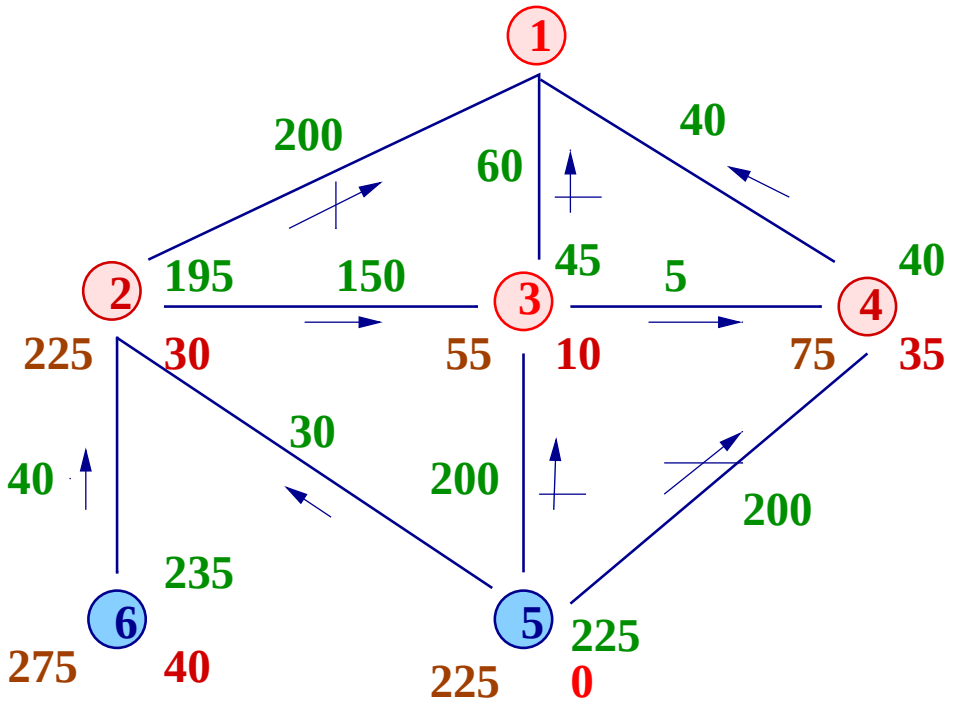
Ejemplo A*



Ejemplo A*



Ejemplo A*


1
200
60
40
2
195
150
3
45
5
4
40
225
30
55
10
75
35
30
40
200
200
235
6
5
225
275
40
225
0

Características

- **Completitud**: si existe solución, la encuentra
- **Admisibilidad**: si hay una solución óptima, la encuentra si
  - el número de sucesores es finito para cada nodo,
  - $k(n_i, n_j) > \epsilon > 0$ en cada arco, y
  - La función $h(n)$ es admisible: $h(n) \leq h^*(n)\ \forall n$

Consideraciones sobre la heurística

- Si $h_1(n) <= h_2(n) \forall n$, $h_2(n)$ está más informada que $h_1(n)$ y servirá para expandir menos nodos
  - Ejemplo: distancia de Manhattan está más informada que número de casillas mal colocadas (problema de Manhattan es menos relajado que el del número de casillas)
- Extremos
  - $h(n) = 0$ para cada nodo: no se tiene información (Dijkstra)
  - $h(n) = h^*(n)$ para cada nodo: se tiene información perfecta
- No tiene sentido dedicar más coste computacional a calcular una buena $h(n)$ que a realizar la búsqueda equivalente: equilibrio

Técnicas mejor-primero

- No informadas:
  - Búsqueda en amplitud: $f(n) = \text{profundidad}(n)$
  - Dijkstra: $f(n) = g(n)$
- Informadas (heurísticas):
  - A, A*, IDA*, ...: $f(n) = g(n) + h(n)$
  - Ponderadas: $f(n) = g(n) + \omega h(n), \omega > 1$
    - Es completo, pero no es admisible
    - La solución óptima tiene un coste menor o igual que $(1 + \omega)$ veces la generada

IDA* (Korf, 1985)

Procedimiento IDA* (Estado-inicial Estado-final)

EXITO=Falso
$\eta = h(s)$
Mientras que EXITO=Falso
    EXITO=Profundidad (Estado-inicial,η)
    $\eta = \min_{i=1,n} \{f(i)\} = \min_{i=1,n} \{g(i) + h(i)\}$
Solución=camino desde nodo del Estado-inicial
        al Estado-final por los punteros

Profundidad (Estado-inicial,η)
    Expande todos los nodos cuyo coste $f(n)$ no excede η

Características

- **Completitud**: es completo
- **Admisibilidad**: es admisible
- **Eficiencia**: la complejidad de tiempo es exponencial, pero la complejidad de espacio es lineal en la profundidad del árbol de búsqueda
- Aunque pudiera parecer lo contrario, el número de *re-expansiones* es sólo mayor en un pequeño factor que el número de expansiones de los algoritmos de el mejor primero
- Fue el primer algoritmo que resolvió óptimamente 100 casos generados aleatoriamente en el 15–Puzzle

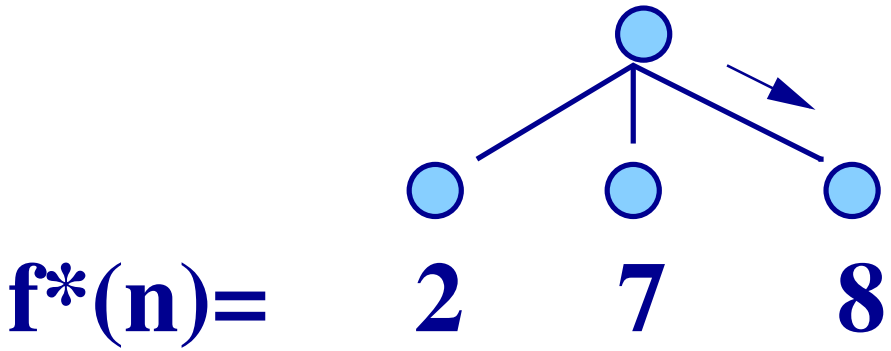
Indice

Características de los problemas

- **Suma nula**: lo que gana uno, lo pierde el otro
- **Dos agentes** (se puede generalizar, Max^n)
- **Información completa**: se conoce en cada momento el estado completo del juego
- **Deterministas** o de **información perfecta**: no entra en juego el azar (se puede generalizar)
- **Alternados**: las decisiones de cada agente se toman de forma alternada

Algoritmo Minimax

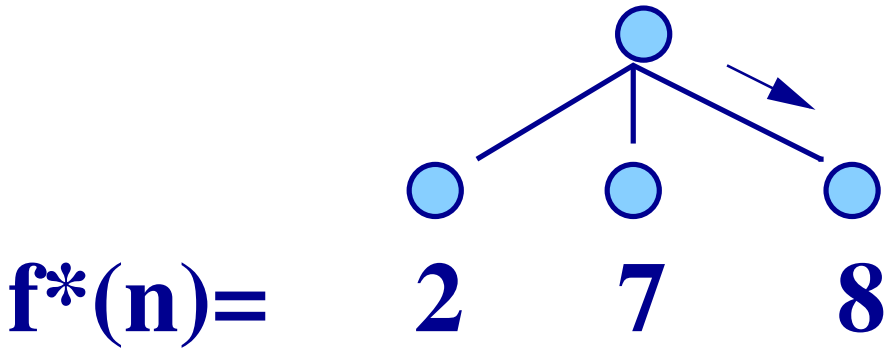
Función de evaluación perfecta



- Problema: no se conoce dicha función

Algoritmo Minimax

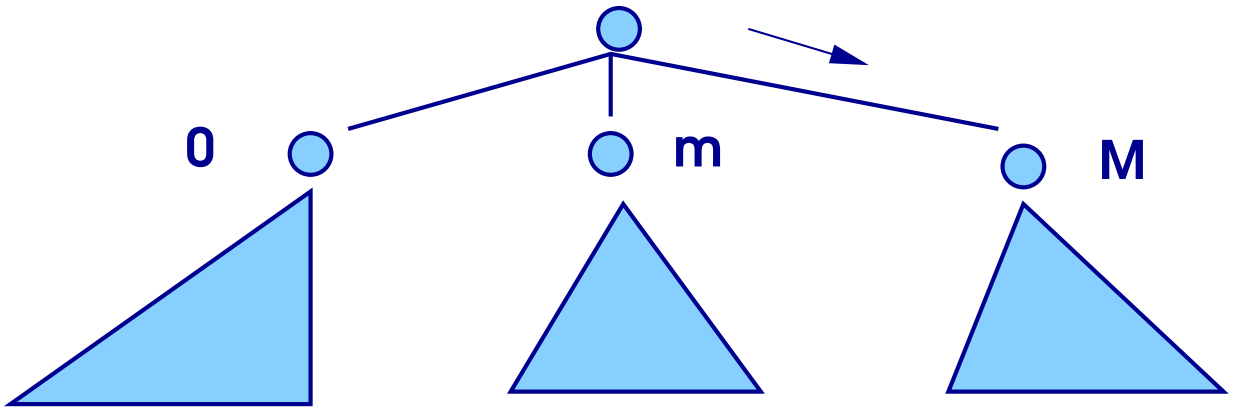
Función de evaluación perfecta



- Problema: no se conoce dicha función

Algoritmo Minimax

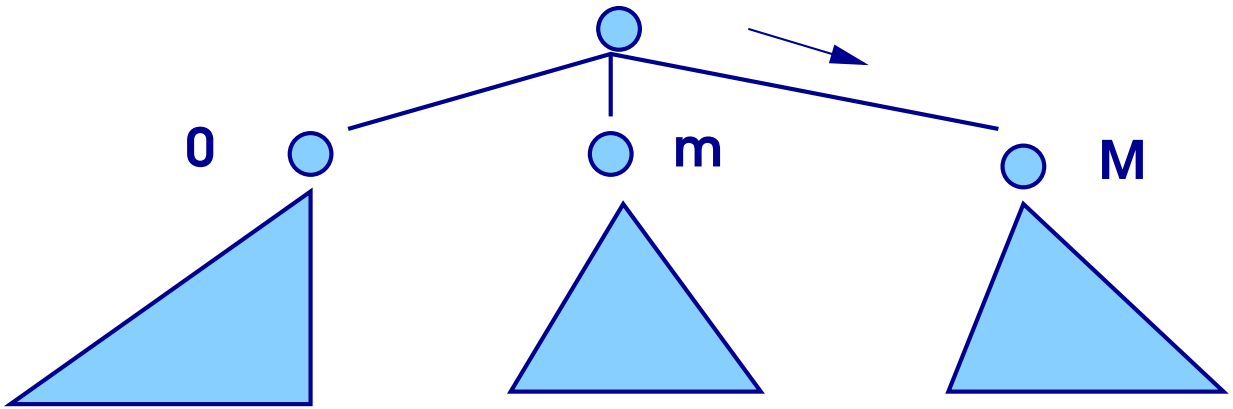
Búsqueda completa



- Nodos hojas: ganamos (M=∞), perdemos (m=$-\infty$) o empatamos (0)
- Problema: es intratable; no se puede realizar en un tiempo razonable

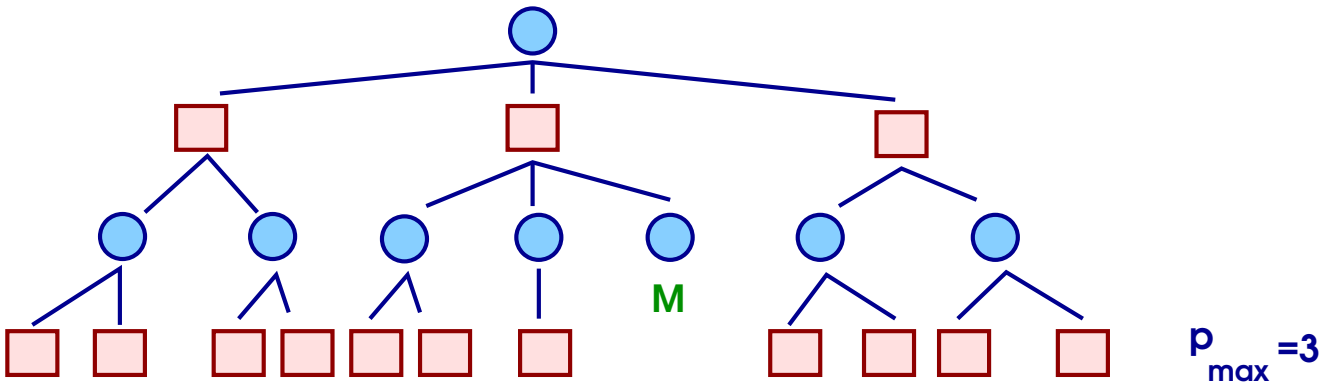
Algoritmo Minimax

Búsqueda completa

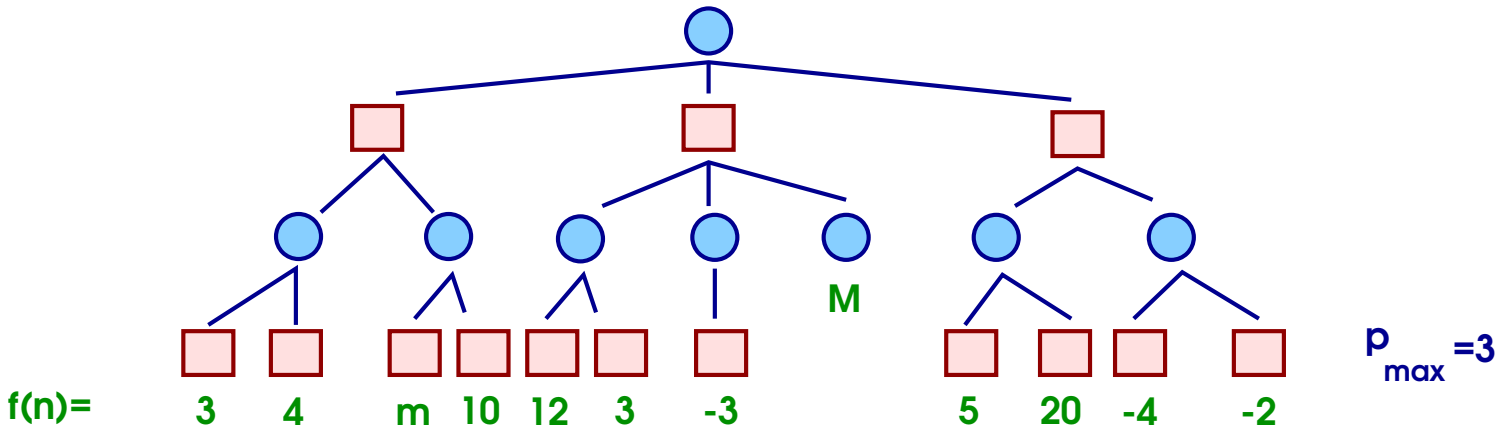


- Nodos hojas: ganamos (M=∞), perdemos (m=$-\infty$) o empatamos (0)
- Problema: es intratable; no se puede realizar en un tiempo razonable

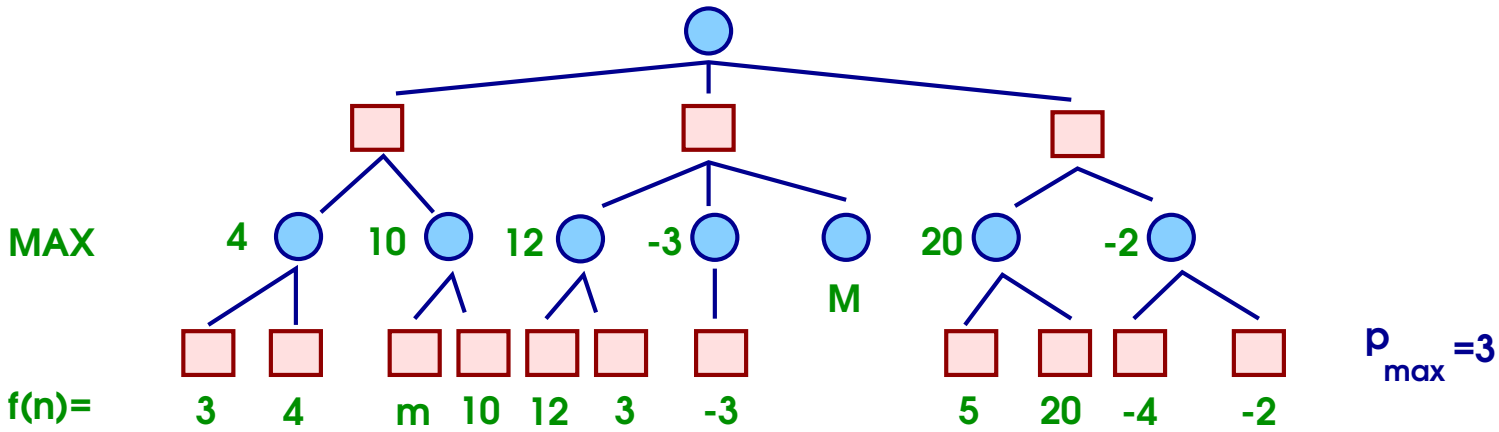
Algoritmo Minimax



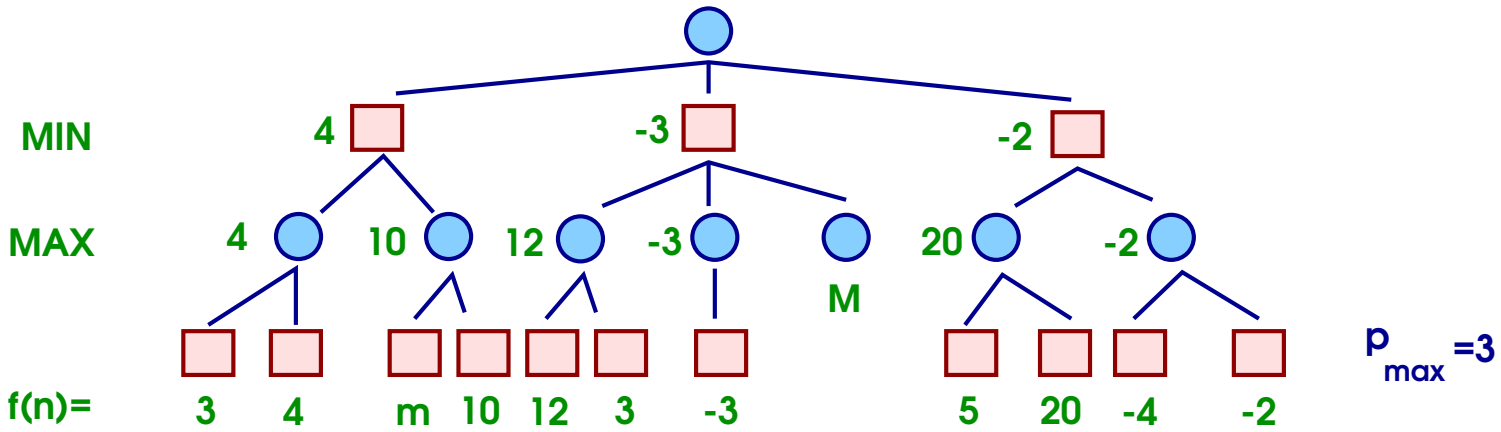
Algoritmo Minimax



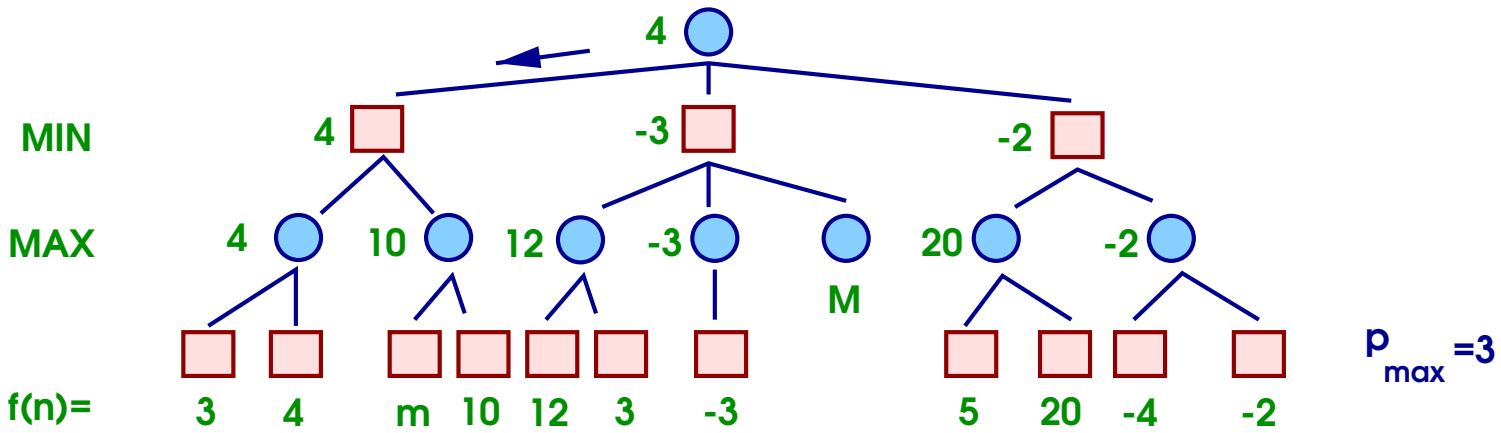
Algoritmo Minimax



Algoritmo Minimax



Algoritmo Minimax



Procedimiento Minimax

Procedimiento Minimax (Situación Profundidad)
SI Profundidad = p_{max}
ENTONCES devolver evaluación (Situación)
SI NO SI ganadora (Situación)
ENTONCES devolver $+\infty$
SI NO SI perdedora (Situación)
ENTONCES devolver $-\infty$
SI NO SI empate (Situación)
ENTONCES devolver 0
SI NO
S = sucesores (Situación)
L = lista de llamadas al MINIMAX ($S_i \in S$, Profundidad + 1)
SI nivel-max (Profundidad)
ENTONCES devolver max (L)
SI NO devolver min (L)

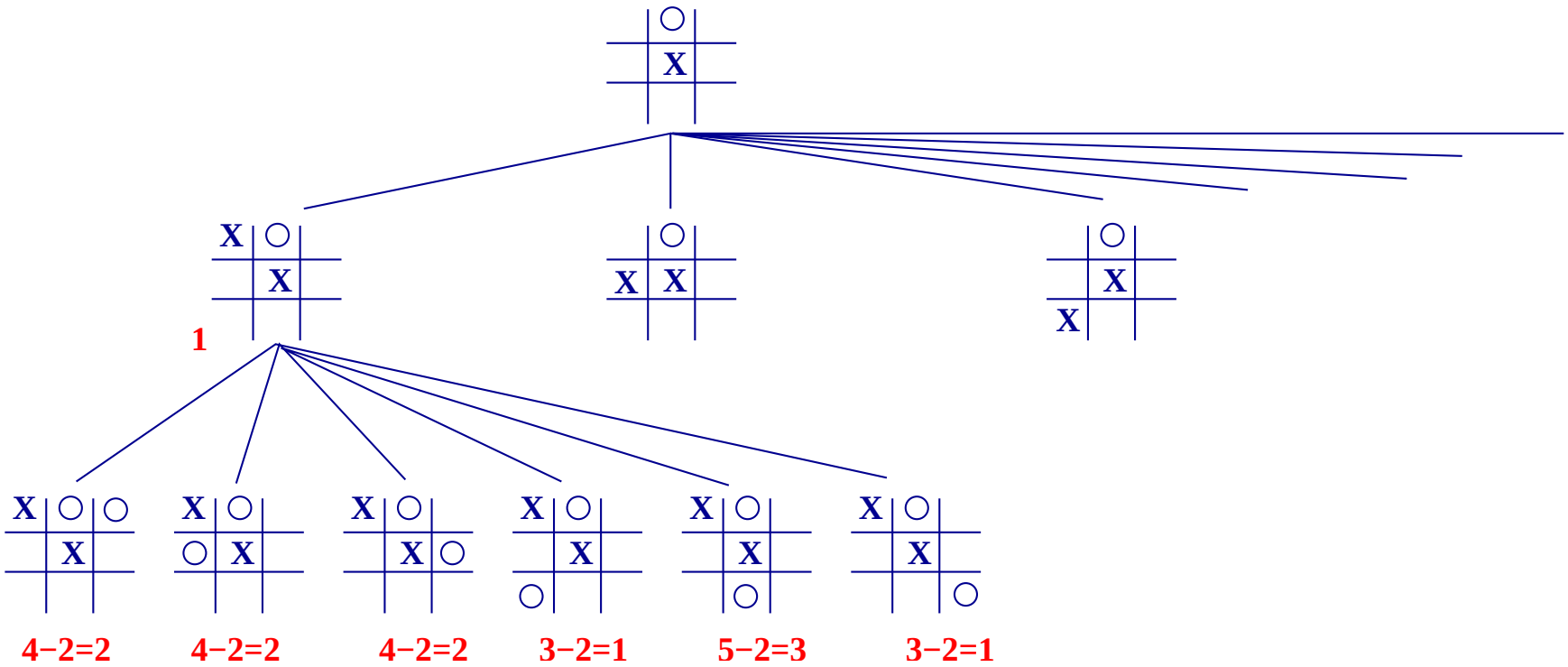
Cálculo del valor de cada nodo

$$f(n) = \begin{cases} +\infty & \text{si n es una situación ganadora para computadora} \\ -\infty & \text{si n es una situación perdedora para computadora} \\ 0 & \text{si n es una situación de empate} \\ f_{ev}(n) & \text{si p = Profundidad máxima} \\ \text{máx}_{S_i \in S(n)} f(S_i) & \text{si n es nodo max y p} < p_{max} \\ \text{mín}_{S_i \in S(n)} f(S_i) & \text{si n es nodo min y p} < p_{max} \end{cases}$$

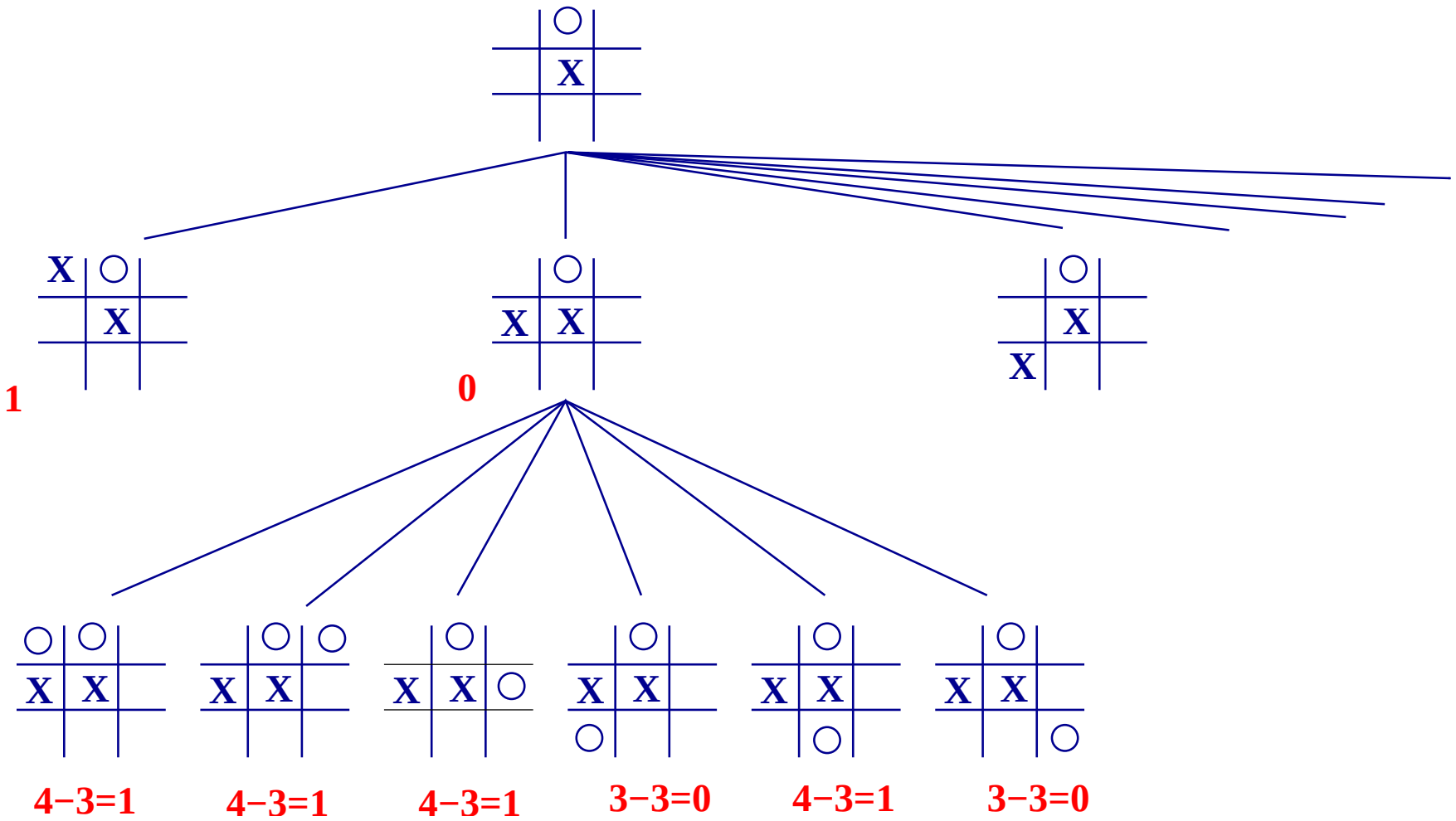
Negamax

$$f(n) = \begin{cases} f_{ev}(n) & \text{si } d = 0 \\ \text{máx}(-f(S_1), ..., -f(S_d)) & \text{si } d > 0 \end{cases}$$

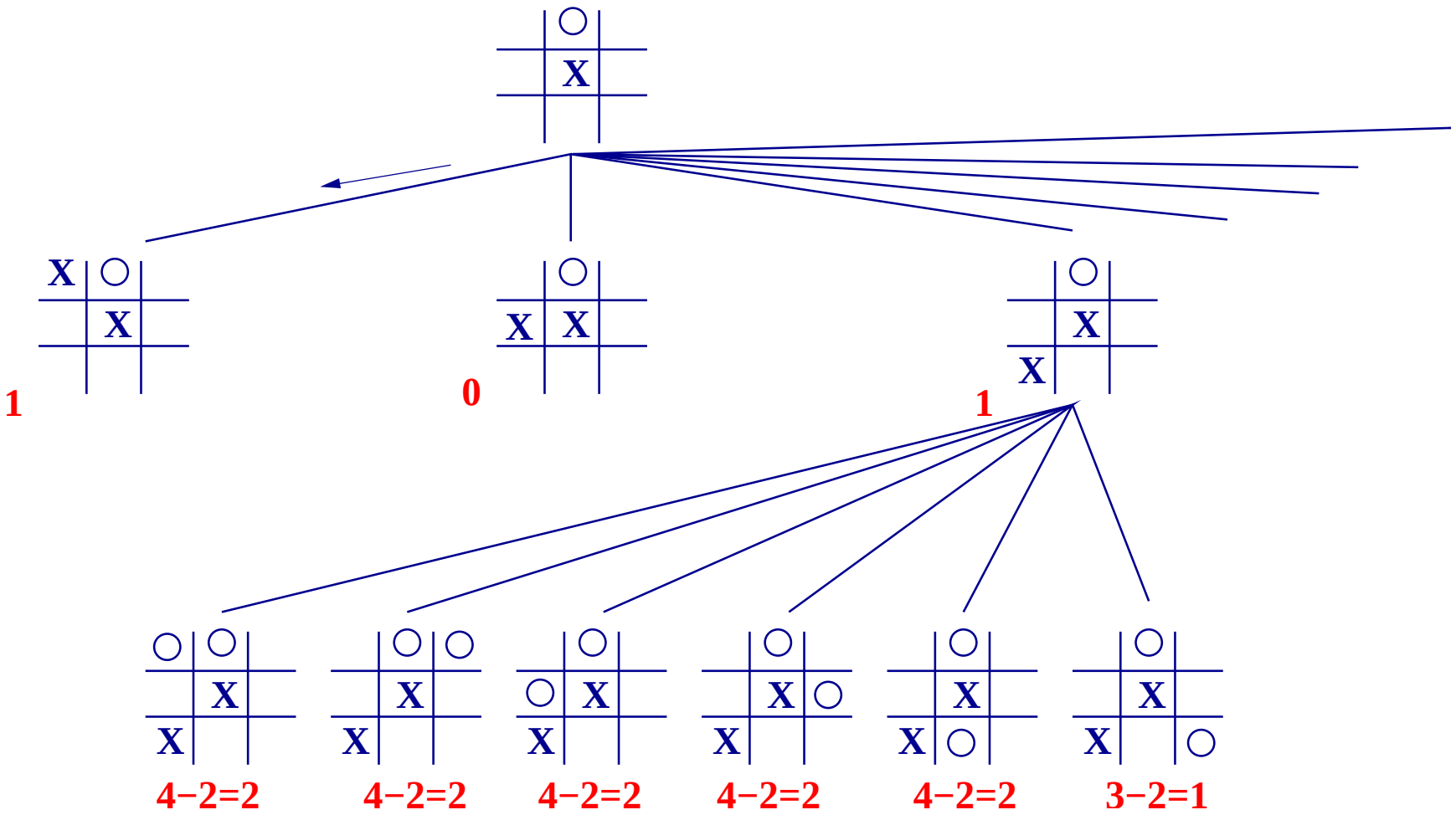
Ejemplo en el tic-tac-toe



Ejemplo en el tic-tac-toe



Ejemplo en el tic-tac-toe



Poda Alfa-beta

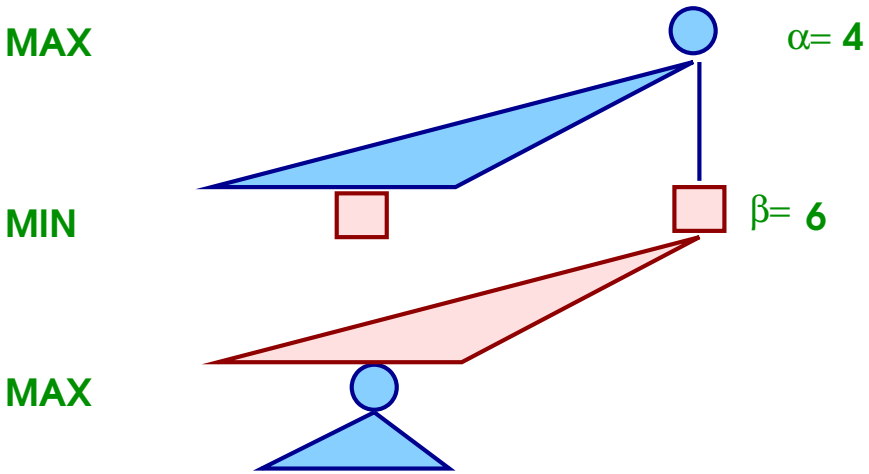
- Nodos MAX: se utiliza el parámetro α para guardar el valor máximo de los sucesores encontrado hasta el momento.
  $\alpha_0 = -\infty$
- Nodos MIN: se utiliza el parámetro β para guardar el valor mínimo de los sucesores encontrado hasta el momento.
  $\beta_0 = +\infty$
- Poda en los nodos MAX:

$$\alpha_p \geq \beta_{p-1}$$

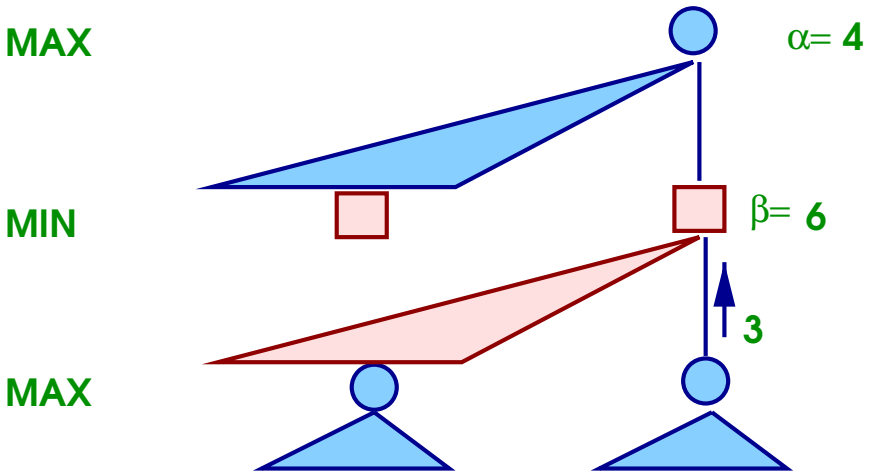
- Poda en los nodos MIN:

$$\beta_p \leq \alpha_{p-1}$$

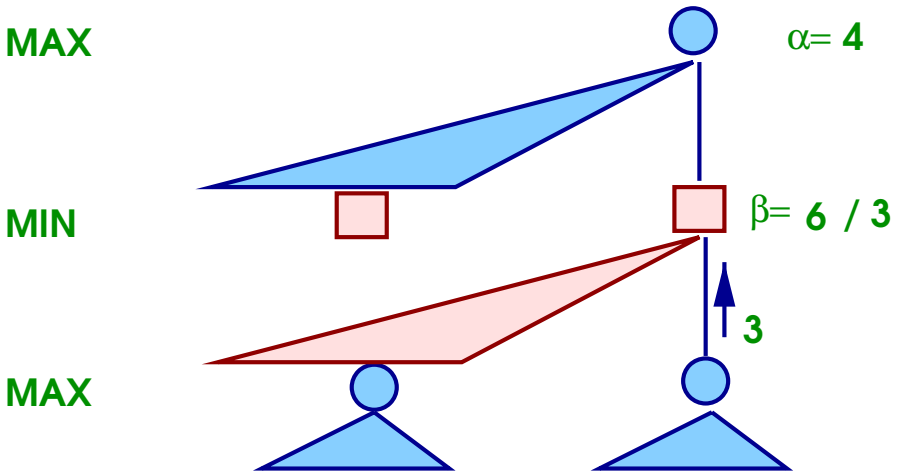
Poda en los nodos MIN



Poda en los nodos MIN

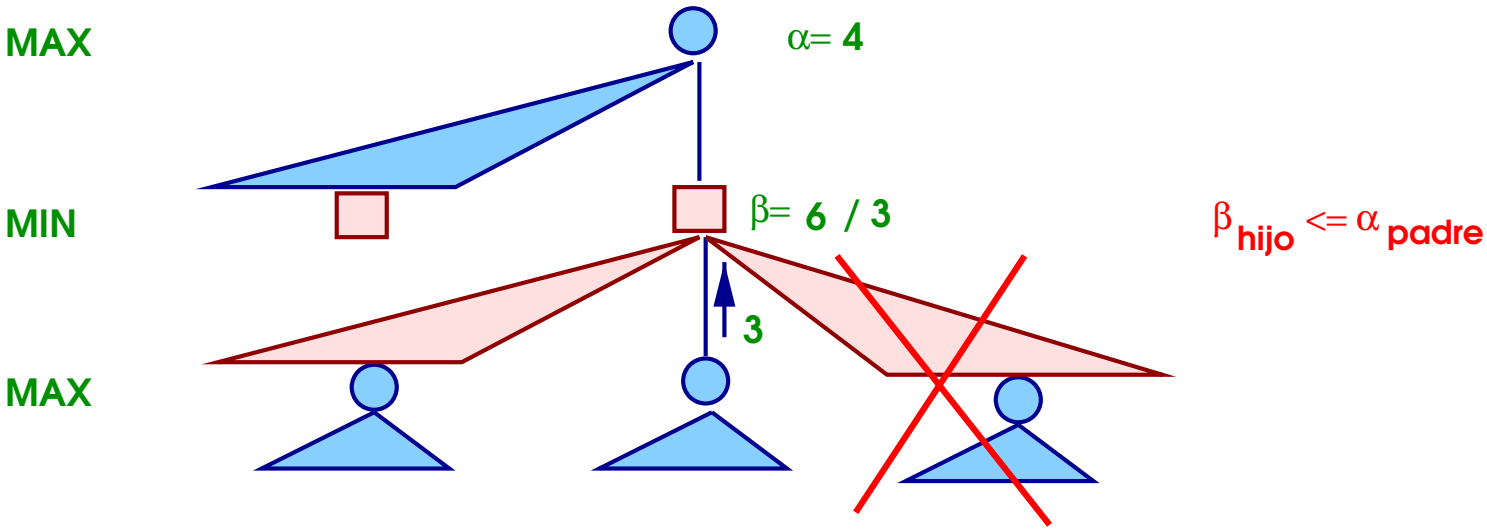


Poda en los nodos MIN

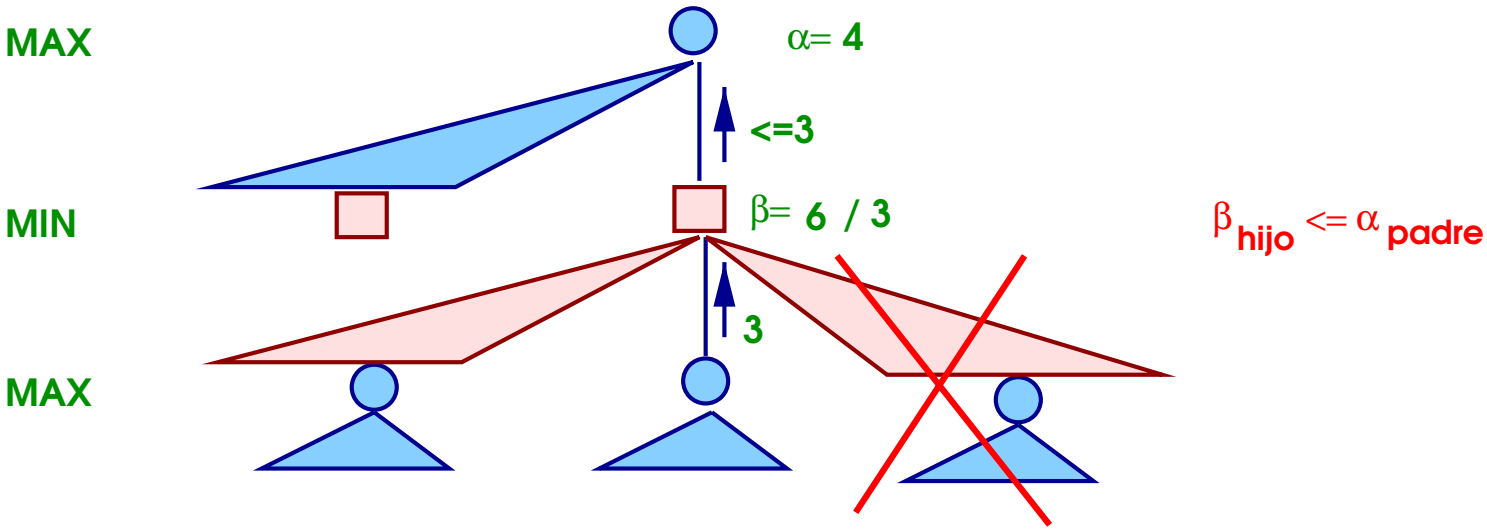


$\beta_{hijo} <= \alpha_{padre}$

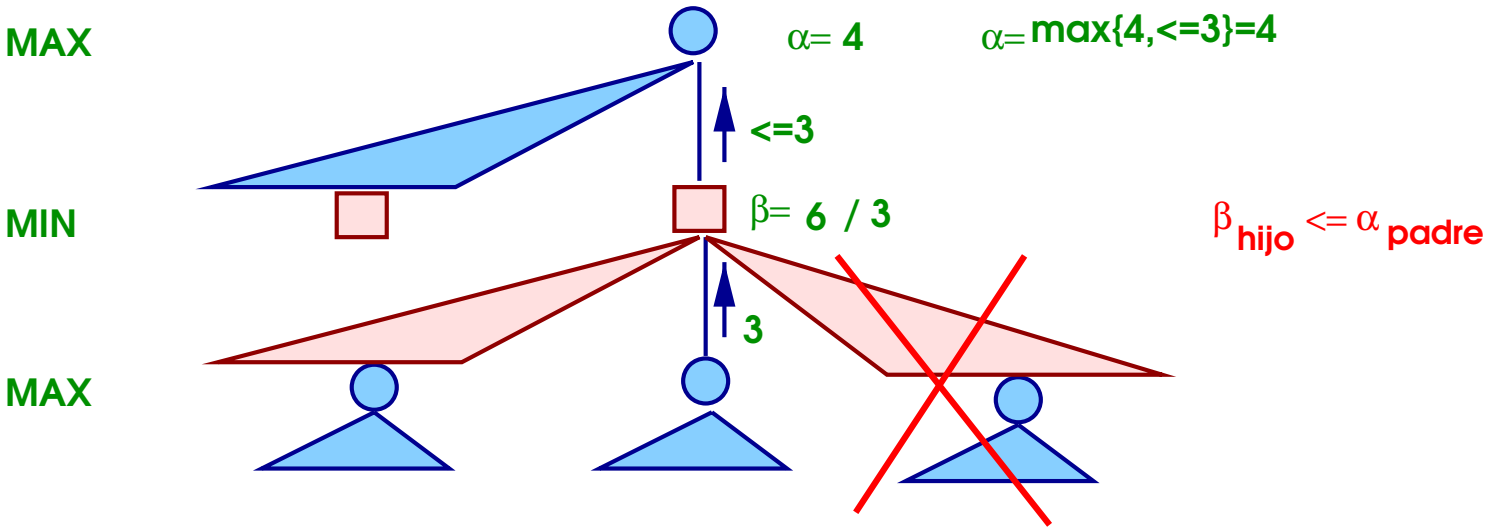
Poda en los nodos MIN



Poda en los nodos MIN



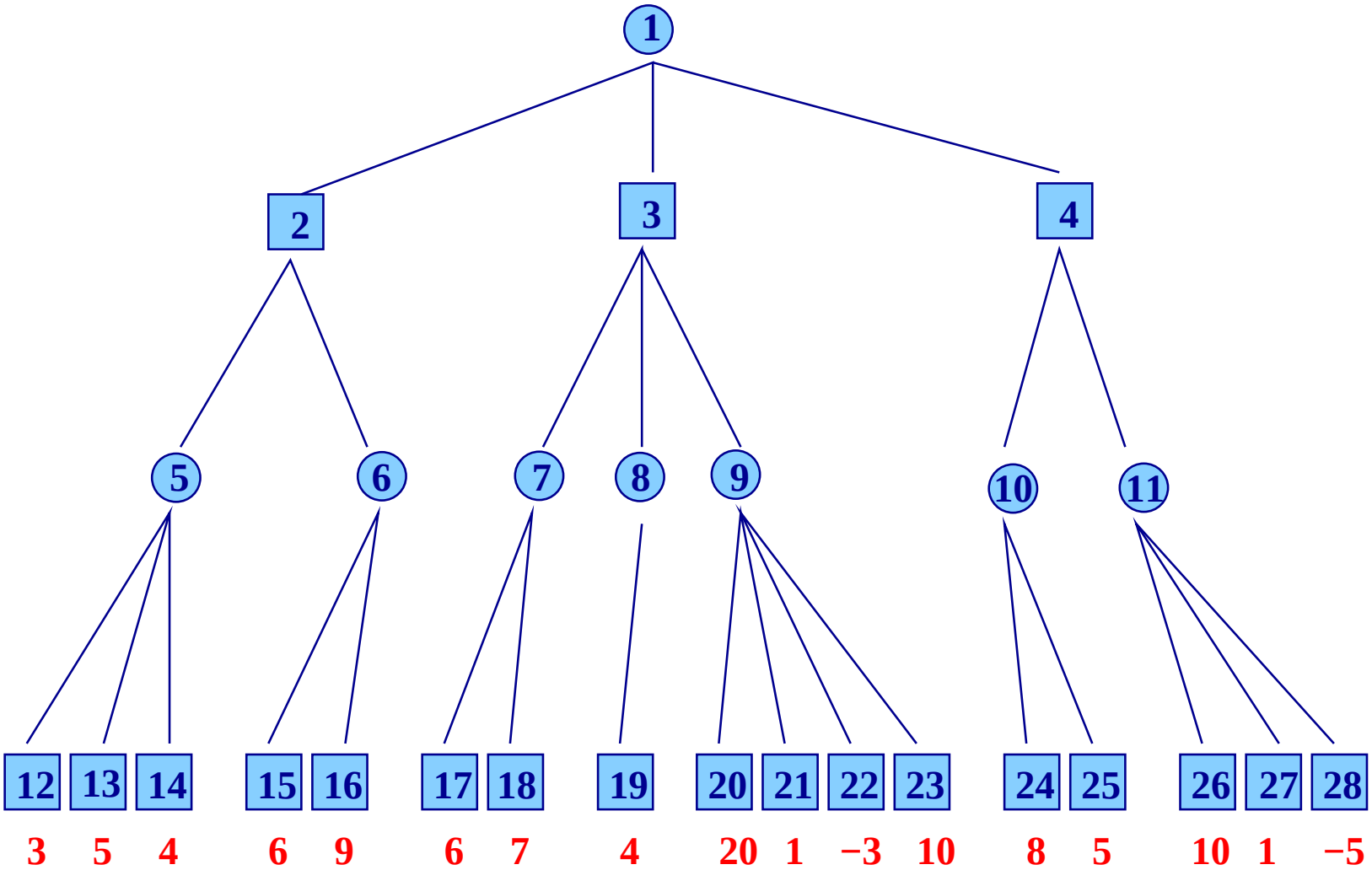
Poda en los nodos MIN



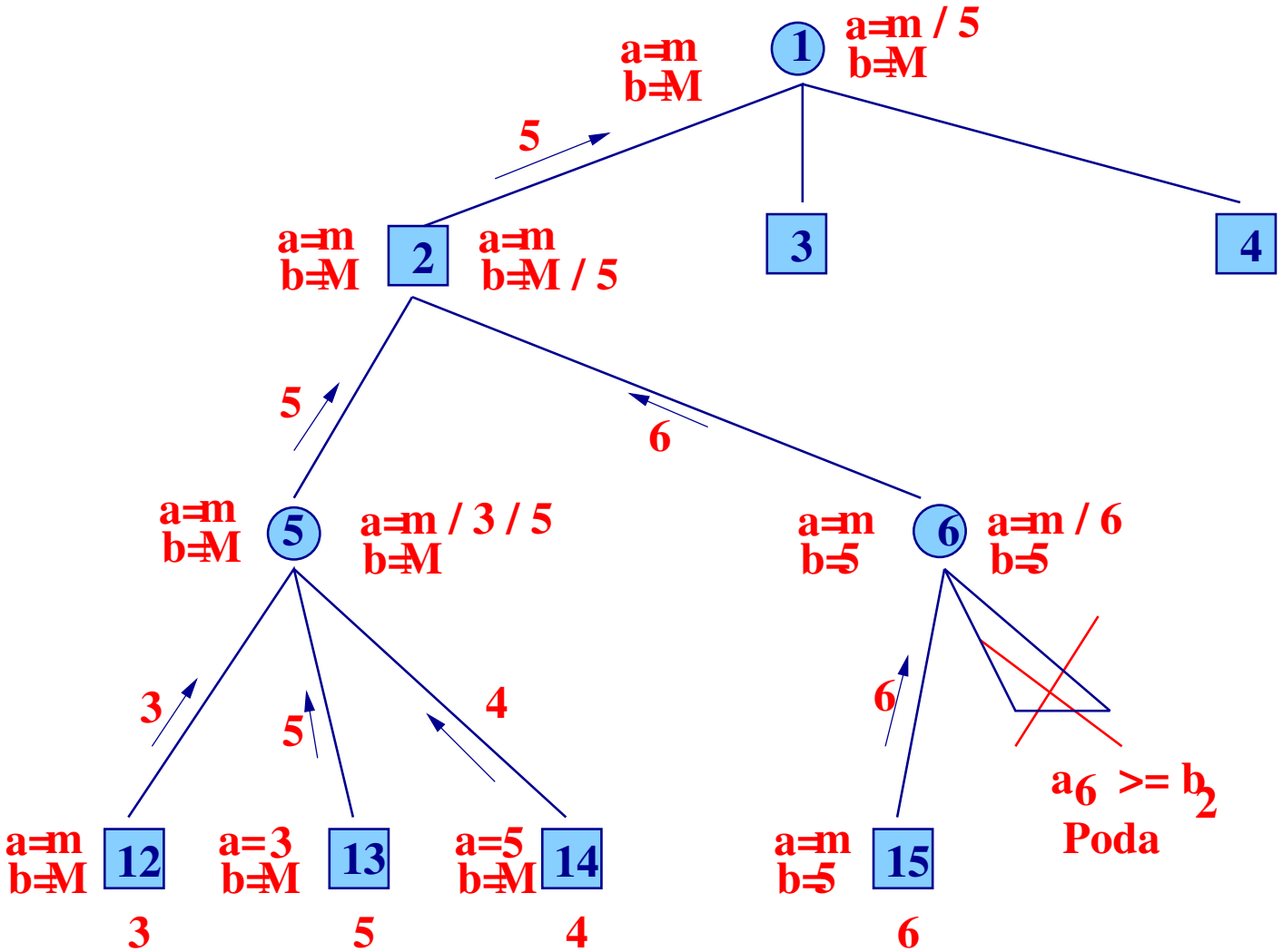
Algoritmo Alfa-beta

Procedimiento Alfa-Beta (Situación Alfa Beta Profundidad)
- SI Situación está considerada como empate, Devolver 0
- SI Situación es ganadora, Devolver mayor-numero
- SI Situación es perdedora, Devolver menor-numero
- SI Profundidad = Profundidad-maxima, Devolver evaluacion (Situación)
- SI nivel-max-p
  - para todo s_i sucesor de Situación y Alfa $<$ Beta
    - Alfa-beta = ALFA-BETA(s_i,Alfa,Beta,Profundidad+1)
    - Alfa = max(Alfa,Alfa-beta)
  - Devolver Alfa
- SI NO
  - para todo s_i sucesor de Situación y Beta $>$ Alfa
    - Alfa-beta = ALFA-BETA(s_i,Alfa,Beta,Profundidad+1)
    - Beta = min(Beta,Alfa-beta)
  - Devolver Beta

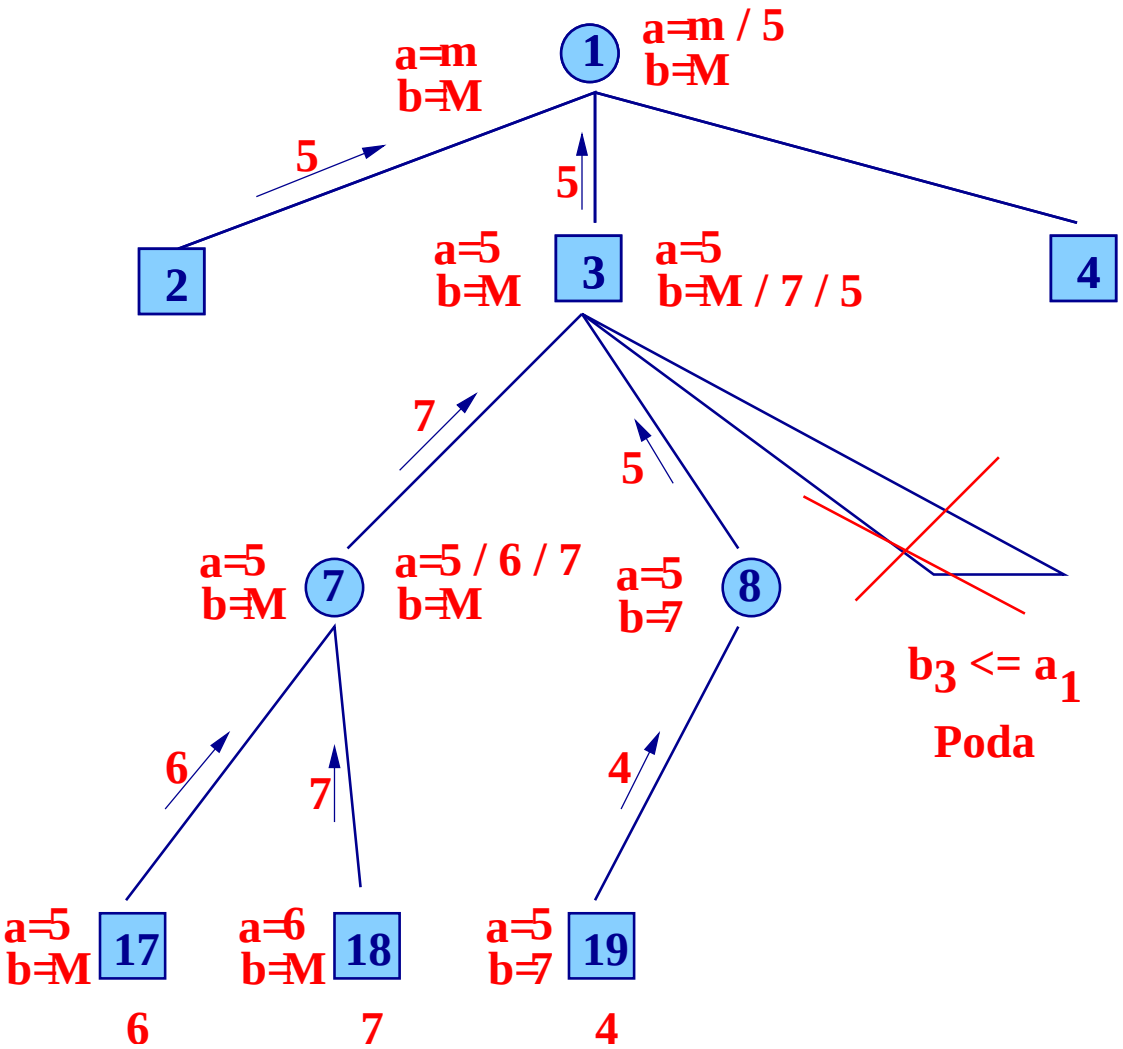
Ejemplo del alfa-beta



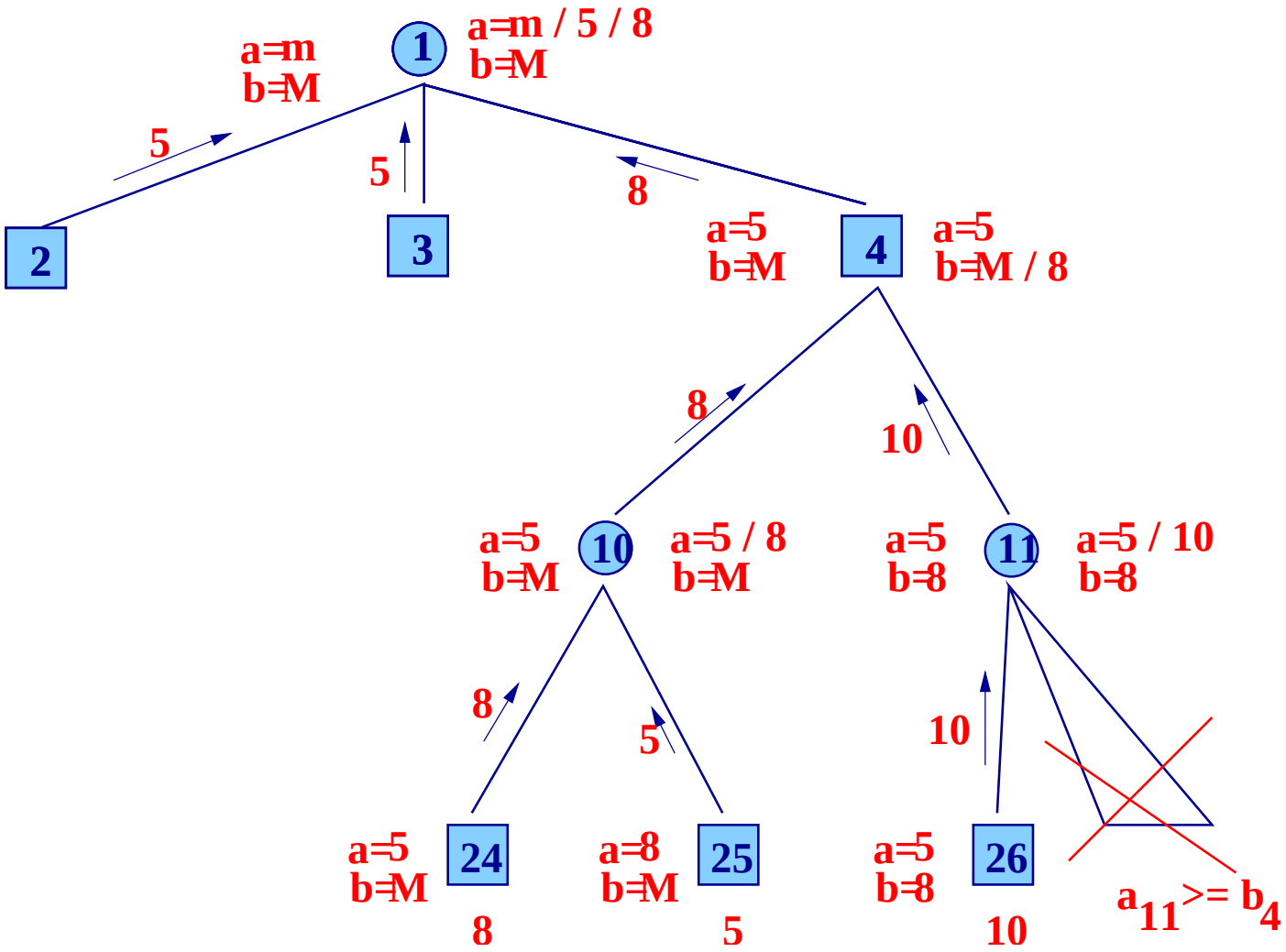
Ejemplo del alfa-beta



Ejemplo del alfa-beta



Ejemplo del alfa-beta



Estado de la cuestión en juegos de dos jugadores

- Ajedrez: HiTech, Deep Thought-Blue, Fritz
- Backgammon: NeuroGammon
- Damas: Chinook
- Othello: Logistello
- Juegos resueltos: Tic-Tac-Toe, Cuatro en Raya, Qubic, Go-Moku, **Damas (oficialmente desde 2007)**, ...
- IA en juegos actuales: estrategia