Inteligencia Artificial

Unidad II: Algoritmos de Búsqueda

ESCOM-IPN Daniel Aguilar Velázquez - Rodolfo Romero Herrera – Gabriela de Jesús López Ruiz

INDICE

- Los problemas resueltos por medio de la búsqueda entre varias alternativas, se basan en la aplicación del sentido común humano. Los humanos generalmente consideran un número de estrategias alternas que las guíen a la solución de problemas. De este modo, se han establecido diferentes alternativas o cursos de acción que conduzcan a la solución en dependencia de las características del espacio de estados del problema a resolver.

2.1 Problemas y espacios de búsqueda

- Los problemas resueltos por medio de la búsqueda entre varias alternativas, se basan en la aplicación del sentido común humano. Los humanos generalmente consideran un número de estrategias alternas que las guíen a la solución de problemas. De este modo, se han establecido diferentes alternativas o cursos de acción que conduzcan a la solución en dependencia de las características del espacio de estados del problema a resolver.

2.1Continuación …

Problemas prácticos

- Búsqueda de rutas
  - Generalización del problema de ir hasta Bucarest.
  - Se añaden complicaciones como el coste en tiempo, dinero, disponibilidad de transporte, etc.
  - Ejemplo: viajes en líneas aéreas.
- Problemas turísticos: visitar ciudades al menos una vez.
- Problema del viajante comercial: viajar a una ciudad sólo una vez con un coste mínimo.
- Distribución de componentes electrónicos en una tablilla de circuito impreso con un mínimo de área.
- Navegación de robots.
- Juegos.
- Búsqueda en internet.

2.1.1 Caracterización de problemas

- El primer paso para solucionar un problema es la **formulación del objetivo**, basado en la situación actual y la medida de rendimiento del agente.

- La tarea del agente es encontrar qué secuencia de acciones permite obtener un estado objetivo.

- Dado un objetivo, la **formulación del problema** es el proceso de decidir qué acciones y estados tenemos que considerar.

- Búsqueda. un agente con distintas opciones inmediatas de valores desconocidos puede decidir qué hacer, examinando las diferentes secuencias posibles de acciones que le conduzcan a estados de valores conocidos, y entonces escoger la mejor secuencia.

- Un algoritmo de búsqueda toma como entrada un problema y devuelve una **solución** de la forma secuencial de acciones. Una vez que encontramos una solución, se procede a ejecutar las acciones que ésta recomienda. Esta es la llamada fase de **ejecución**.

2.1.1 Caracterización de problemas (cont.)

Medidas de rendimiento.

- Completitud: si existe la solución ¿la encuentra?
- Optimización: si la encuentra ¿es la mejor?
- Complejidad espacial-temporal: ¿cuánto tiempo y memoria se necesita para encontrar la solución?
  - En informática teórica se expresa mediante el tamaño del grafo.
  - En IA la dificultad del problema se expresa en función de:
  - Factor de ramificación: máximo número de sucesos por nodo.
  - Profundidad del objetivo: profundidad del objetivo menos profundo.
  - Profundidad máxima en el espacio de estados.
  - ¿Qué cantidad de recursos son necesarios para encontrar la solución? (nodos almacenados en memoria)
  - ¿cuánto se tarda en encontrar la solución? (número de nodos generados).

2.1.2 Espacios de búsqueda

- *Espacio de búsqueda:* Conjunto de estados que podrían obtenerse si se aplicaran todos los operadores posibles a todos los estados que se fuesen generando.

- Cada estado representa una situación que se debe considerar como candidata a pertenecer a solución del problema.

- Los ejemplos más característicos de esta categoría de problemas son los juegos (son universos restringidos fáciles de modelar).

- El espacio de estados de un juego es un grafo cuyos nodos representan las configuraciones alcanzables (los estados válidos) y cuyos arcos explicitan las movidas posibles (las transiciones de estado).

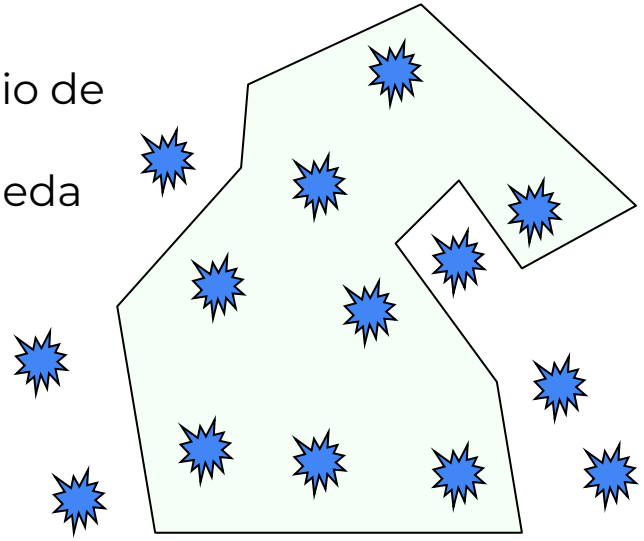
2.1.2 Espacios de búsqueda (cont.)

Al Definir el problema, por tanto se define el espacio de búsqueda.

Los problemas prácticos tienen espacios de búsqueda grandes.

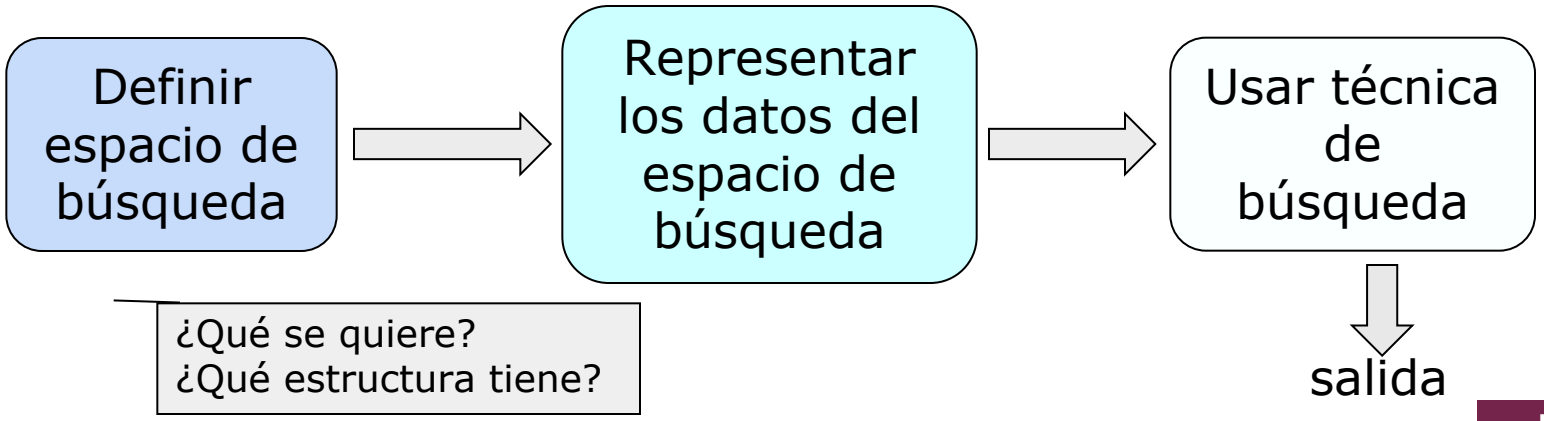
No se pueden tener construcciones de grafos explícitos.

La búsqueda puede ser utilizada para planear secuencias de acciones.



Espacio de búsqueda

Para realizar una búsqueda:



2.1.2 Espacio de búsqueda (cont.)

Terminología:

Estado inicial, punto de partida de la búsqueda.

Función sucesor, dado un estado x devuelve un conjunto ordenado de pares (acción, sucesor)

Espacio de búsqueda, conjunto de estados alcanzables a partir del estado inicial.

Camino, es una secuencia de estados conectados por una secuencia de acciones.

Coste del camino, es una función que asigna un coste numérico a cada camino. Suele ser la suma de costes de cada acción individual.

Prueba objetivo, determina si un estado es el objetivo.

Solución, secuencia de acciones que llevan del estado inicial a un estado objetivo. La que tenga coste del camino mínimo, serán óptimas.

Ejercicio (cont.)

Aplicaciones
Rompecabezas numérico

Estados: especifican la posición de cada ficha.
Estado inicial: una posición.
Función sucesor: genera los estados resultantes de "desplazar el hueco", arriba, abajo, derecha e izquierda.
Prueba objetivo: indica si la combinación de fichas es la deseada.
Coste: número de movimientos.

complejidad y combinaciones (n+1)!
8 piezas: 362,880
15 piezas: $2.1X10^{13}$
24 piezas: $1.5X10^{25}$

2	8	3
1	6	4
7		5

Estado inicial

1	2	3
8		4
7	6	5

Estado objetivo

2.2 Algoritmos de búsqueda no informados

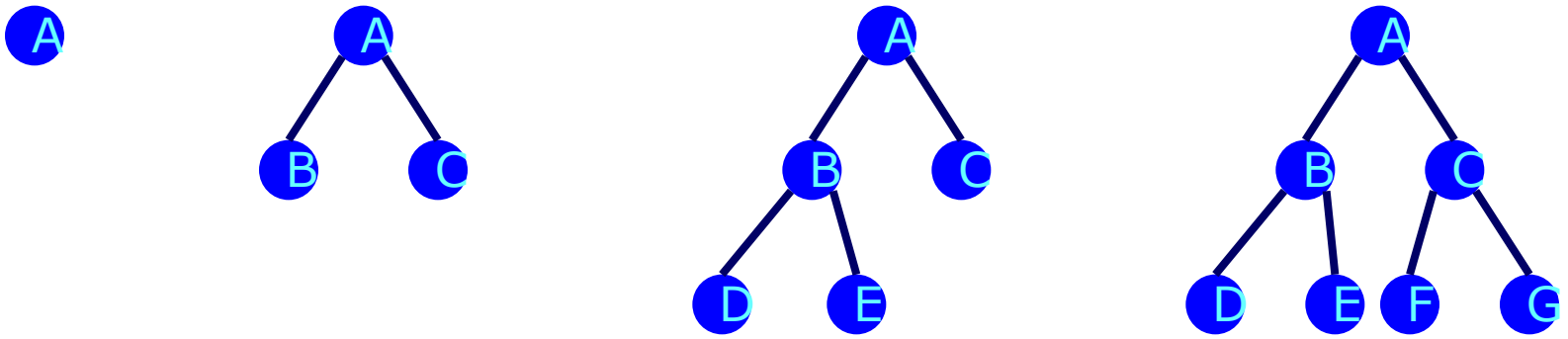
Técnicas de búsqueda y su grado de conocimiento:

Búsqueda a ciegas.

- Búsqueda primero en profundidad.
- Búsqueda primero en anchura.

2.2.1 Búsqueda en anchura

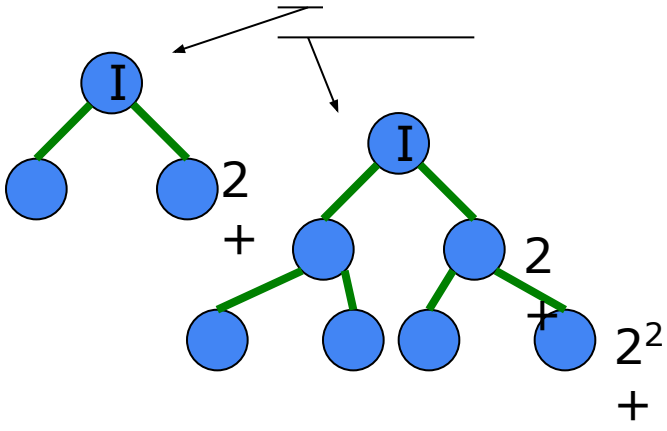
- Está técnica hace uso de un árbol no necesariamente binario.
- El árbol se crea con base a las reglas de producción definidas en el problema.
- Cada nodo almacena un estado del espacio de estados del problema.
- En general la búsqueda primero en anchura consta de los siguientes pasos: Se expande el nodo raíz, a continuación se expanden los sucesores, y luego los sucesores de estos. . .

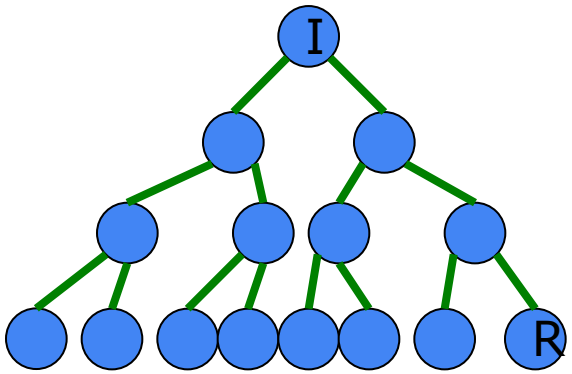


2.2.1 Búsqueda en anchura (cont.)

- Completitud: es una técnica completa si el factor de ramificación es finito.
- Optimización: no se garantiza una solución óptima.
- Complejidad: si cada nodo tiene *b* sucesores y la solución está a nivel *d*, en el peor de los casos se tiene que expandir todos menos el último nodo del nivel d

$$b + b^2 + b^3 + \ldots + b^d = \sum_{i=1}^{d} b^i$$





O(b^d)
Tiempo
espacio

$2+2^2+2^3=14$

2.2.1 Búsqueda en anchura (cont.)

Ejemplo del rompecabezas numérico.

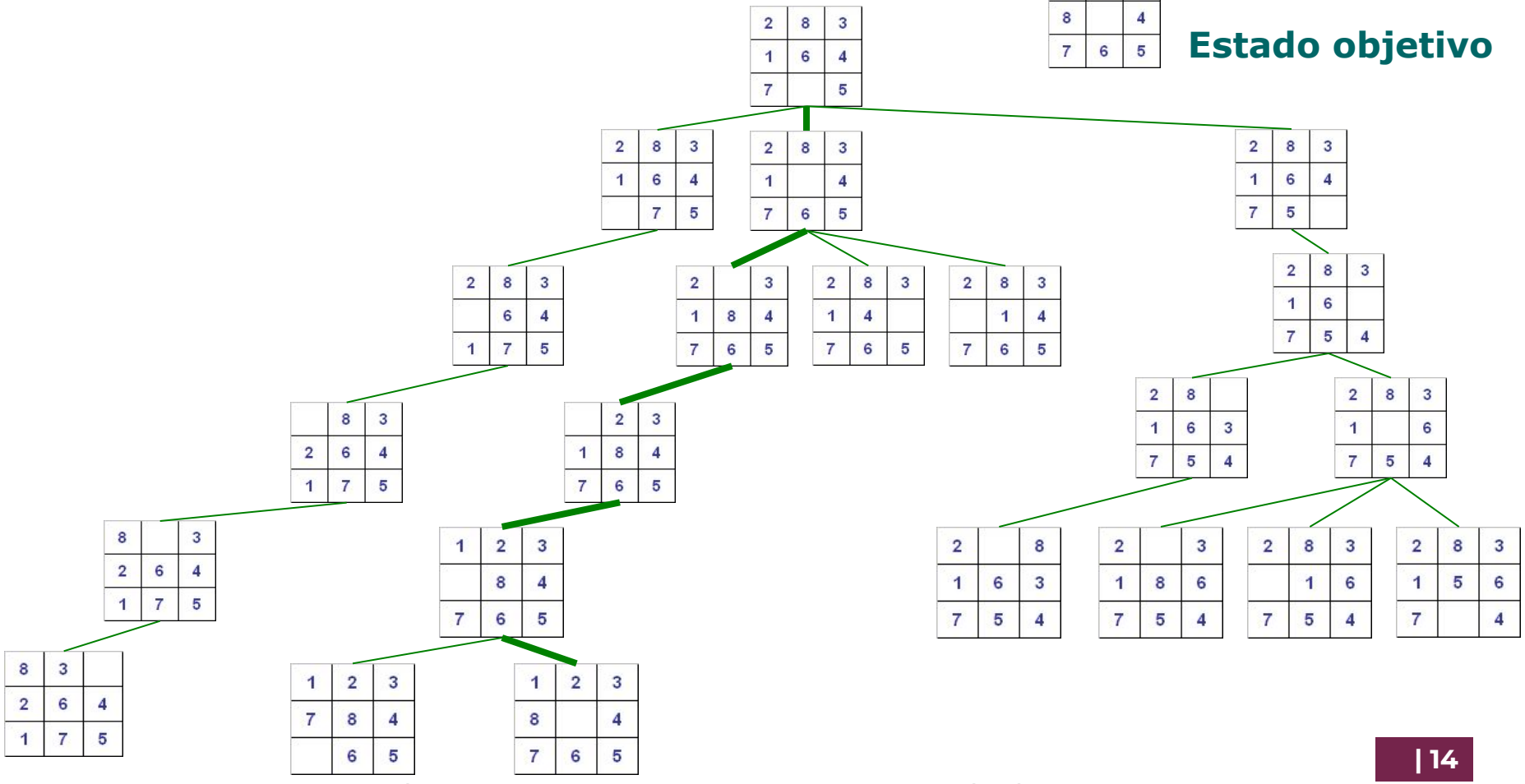
Reglas: mover la celda vacía a la izquierda, arriba, a la derecha y abajo. Siendo movimientos reversibles.

2	8	3
1	6	4
7		5

Estado inicial

1	2	3
8		4
7	6	5

Estado objetivo



2.2.1 Búsqueda en profundidad

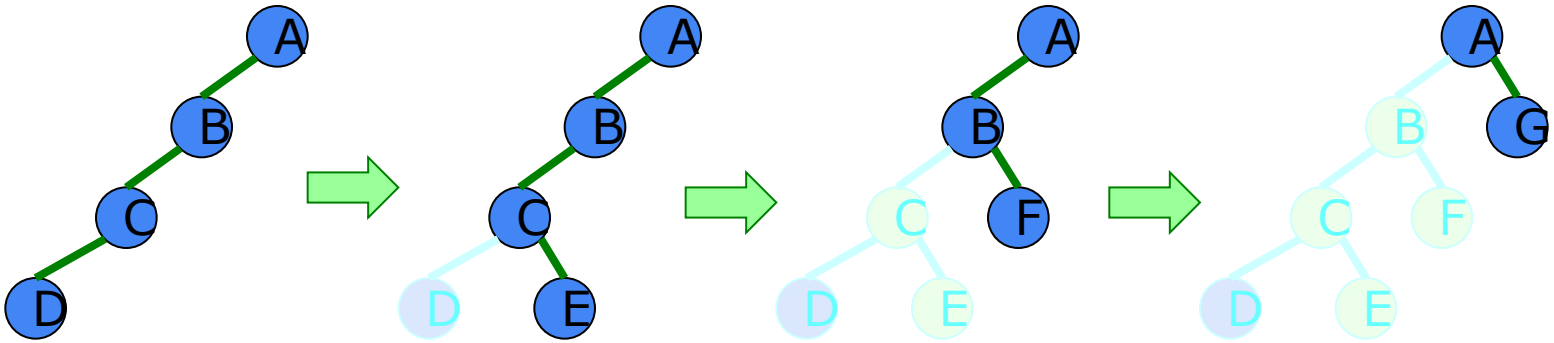
Funcionamiento general

Se expande en profundidad un sólo nodo del árbol.

La expansión se interrumpe si se condiciona a un límite finito de niveles de expansión o cuando el nodo no posee más sucesores.

Cuando sucede la interrupción, la búsqueda retrocede al nivel anterior.

Suponiendo que se pone un límite de 3 niveles:



Suponiendo que no se encontró el objetivo

Sin encontrar el objetivo en E, y C sin tener más sucesores

Sin encontrar el objetivo en F, y B sin tener más sucesores

Si G es el objetivo, entonces la solución son los nodos que se encuentran del inicial a la solución

2.2.2 Búsqueda en profundidad (cont.)

Completitud: si hay ramas infinitas el proceso de búsqueda podría no terminar, aún teniendo una solución próxima a la raíz.
Optimización: no se garantiza que la solución sea óptima.
Complejidad temporal: si *d* es la profundidad máxima del árbol y *b* el número de sucesores que se crean en el árbol:

$$b + b^2 + b^3 + \ldots + b^d = \sum_{i=1}^{d} b^i$$

O(b^d) complejidad exponencial

Complejidad espacial: siendo *m* la profundidad donde se encontró el objetivo y *b* el número de sucesores que pertenecen a la solución:

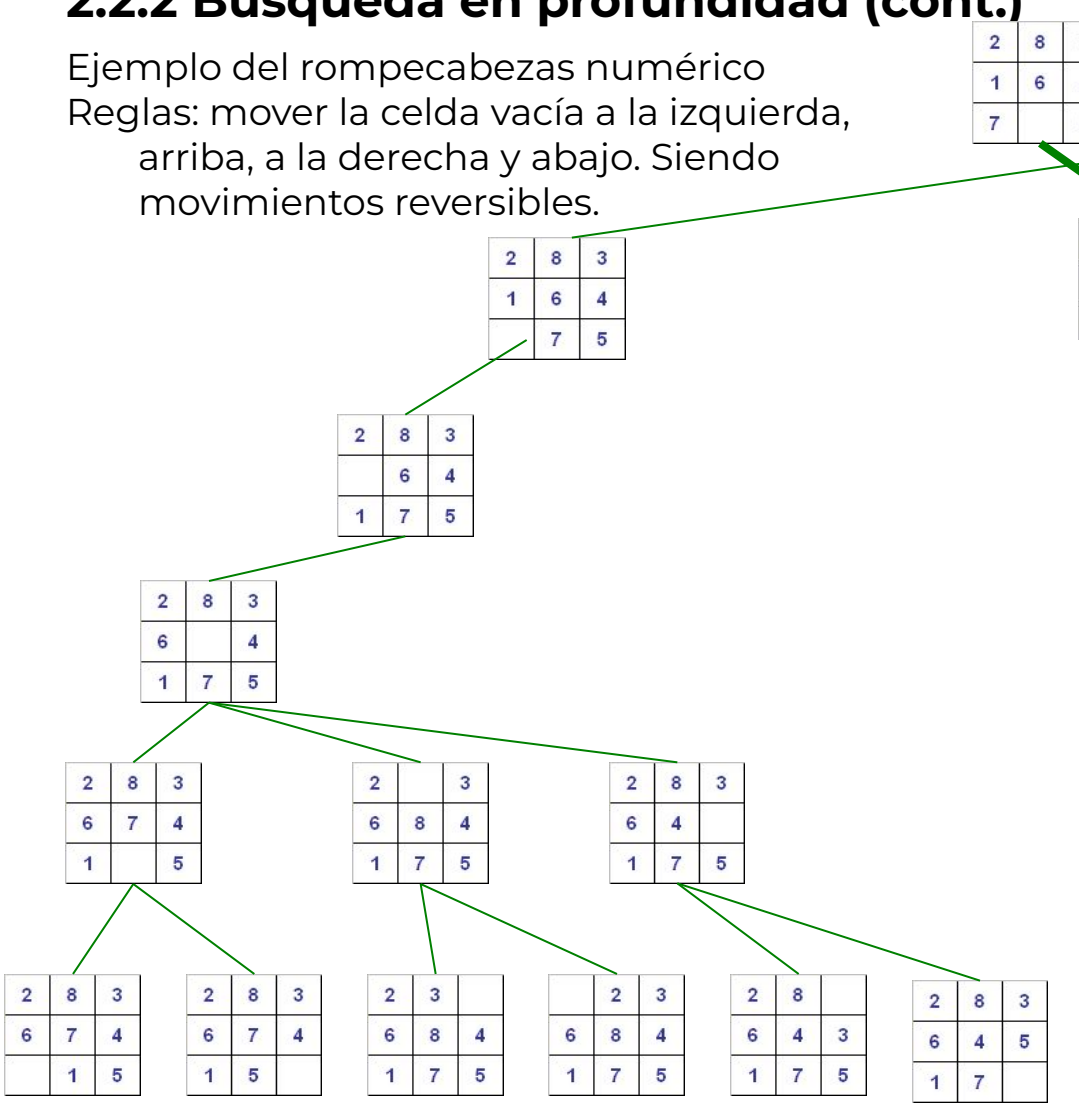
$$m + 1$$

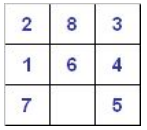
O(m) Complejidad lineal

2.2.2 Búsqueda en profundidad (cont.)

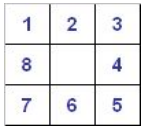
Ejemplo del rompecabezas numérico

Reglas: mover la celda vacía a la izquierda, arriba, a la derecha y abajo. Siendo movimientos reversibles.





Estado inicial



Estado objetivo

El límite se establece a 5 niveles

2.3 Algoritmos de búsqueda informados

2.3.1 Heurística
2.3.2 Algoritmo de escalada simple y primero el mejor
2.3.3 Algoritmo A*

2.3.1 Heurística

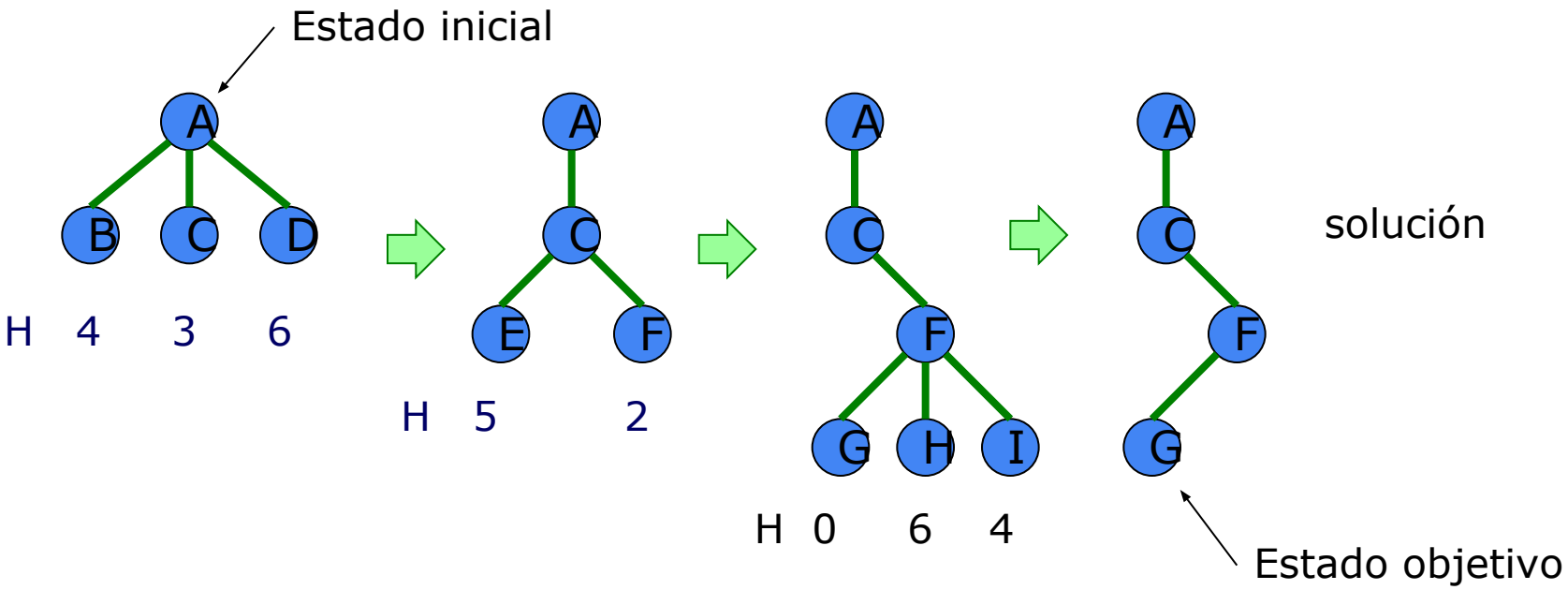
? **Búsqueda heurística.**
- Búsqueda en escalada.
- Búsqueda el primero mejor
- Algoritmo A*

Heurística. En algunas ciencias, la manera de buscar la solución de un problema mediante métodos no rigurosos, como por tanteo, reglas empíricas, etc.

La función heurística, que llamaremos h(n), trata de guiar la búsqueda para llegar de forma más rápida a la solución. A partir de la información disponible, la función heurística intenta estimar el coste del mejor camino (menor coste) desde el nodo n hasta el nodo objetivo.

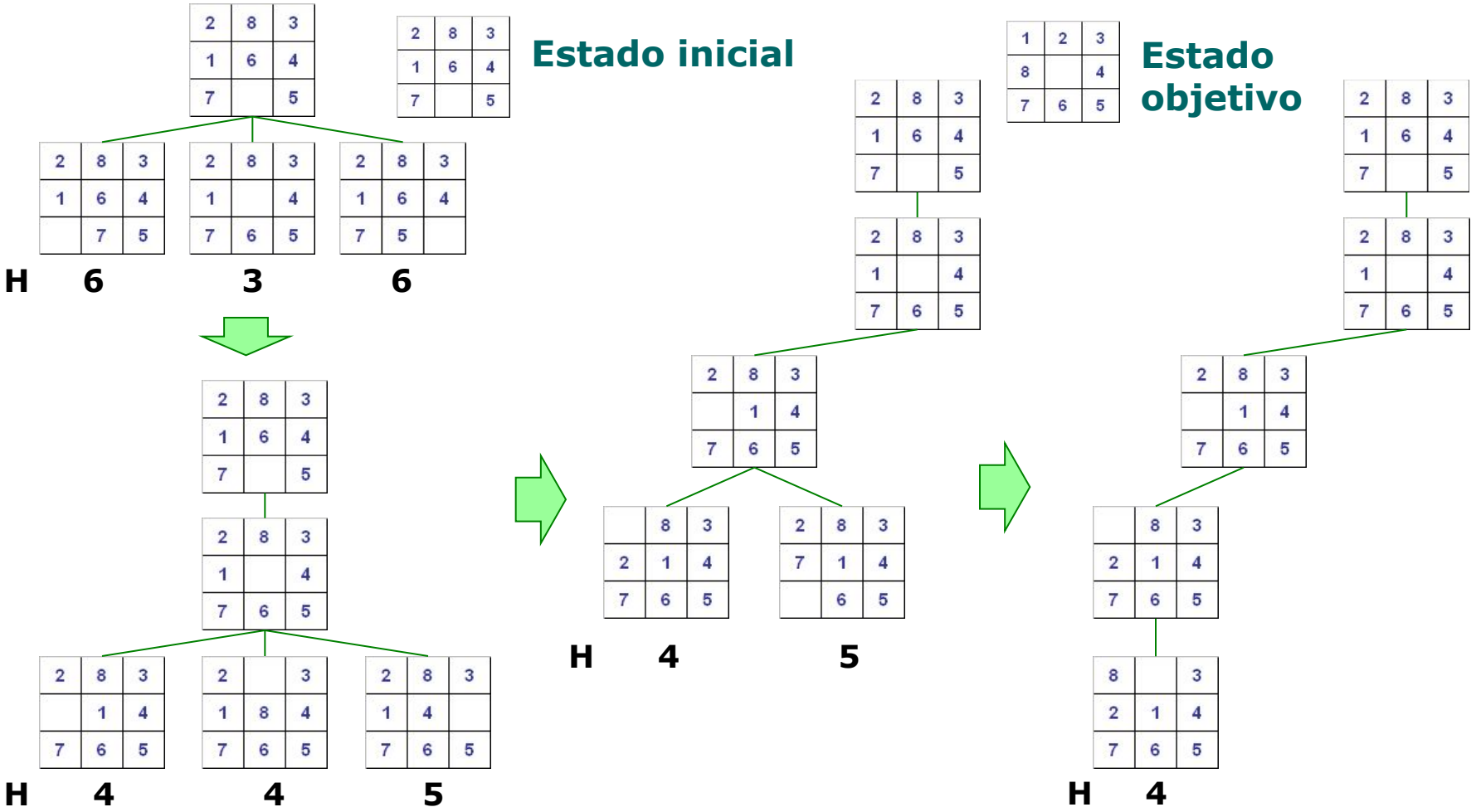
Búsqueda en escalada simple y primero el mejor

- Algoritmo heurístico, e irreversible (voraz) debido a que no se guardan árboles.
- Mide la proximidad de un estado a los objetivos (conocimiento).
- No se garantiza una solución optima.
- Se pueden encontrar soluciones rápidamente.
- Se usa una función sucesor que retorna la proximidad a la solución.

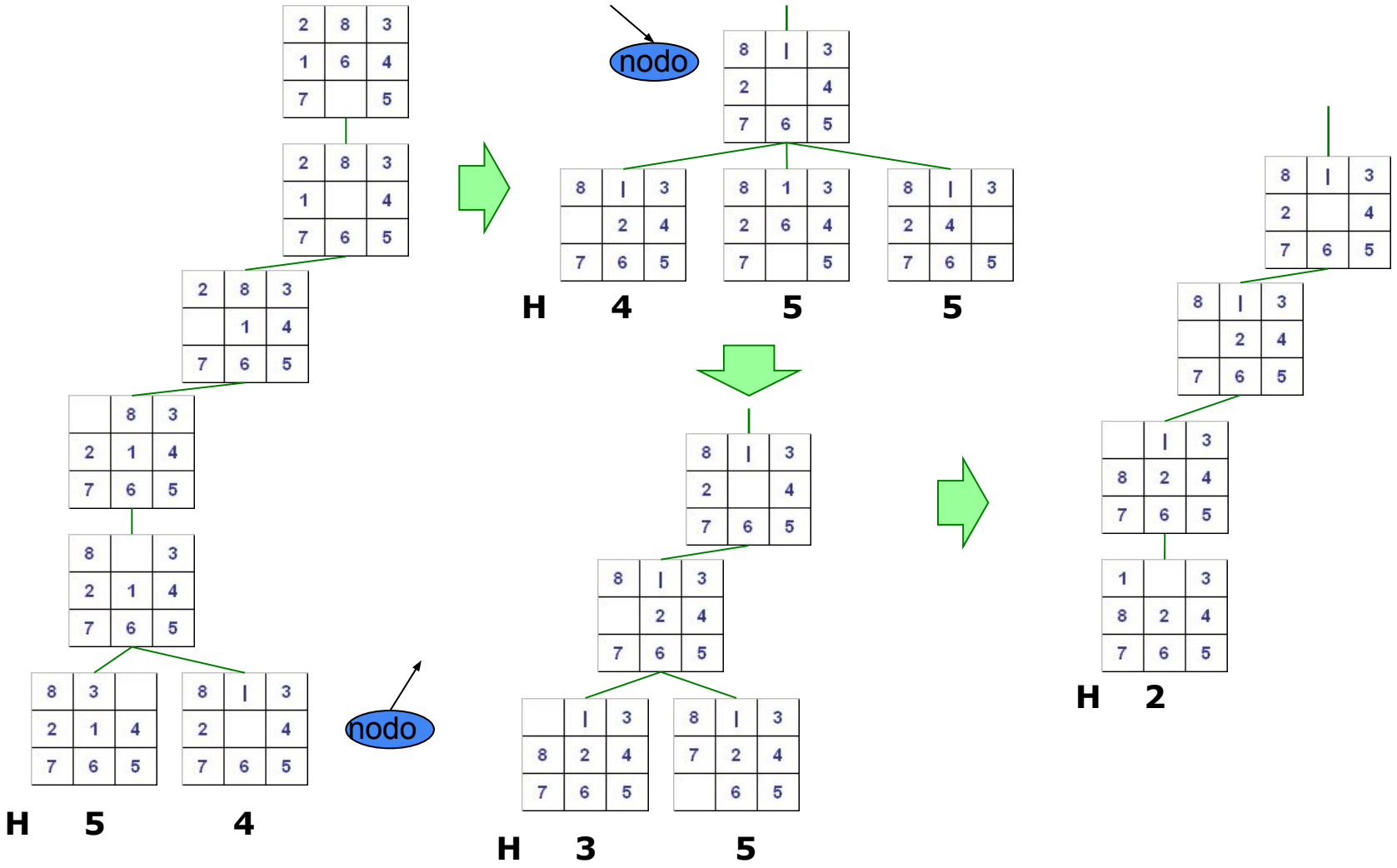


Búsqueda en escalada simple y primero el mejor (cont.)

Ejemplo del rompecabezas numérico

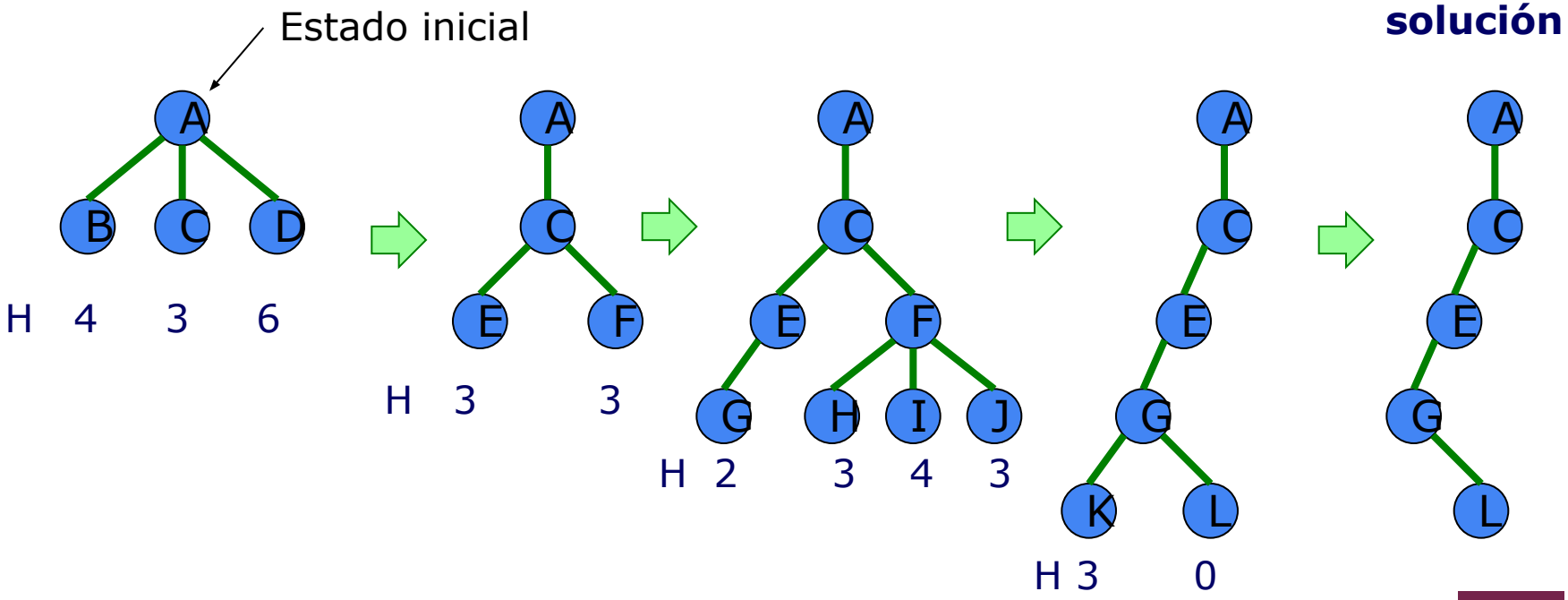


Búsqueda en escalada simple y primero el mejor (cont.)



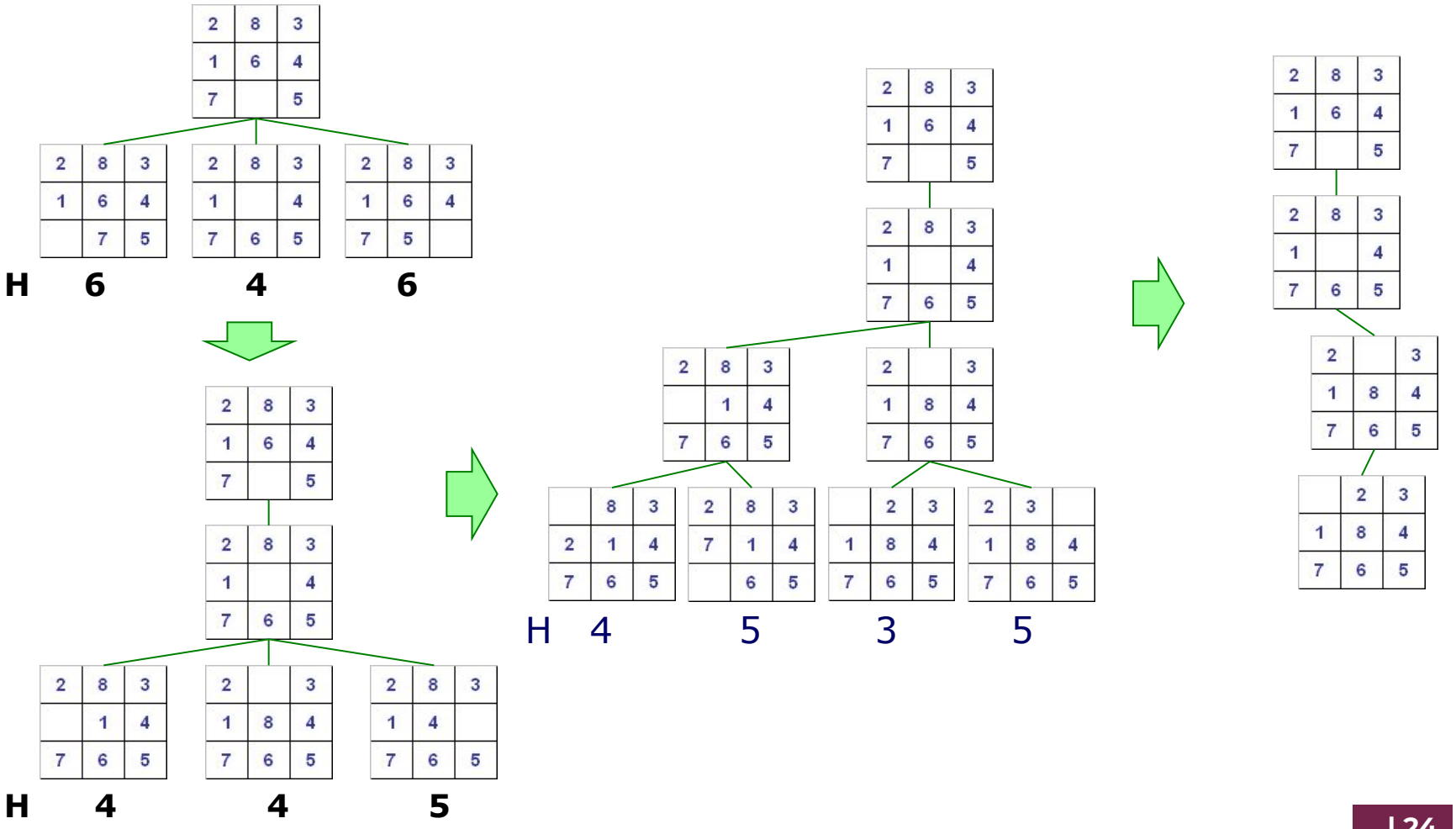
Búsqueda escalada simple y primero el mejor

- Algoritmo heurístico o voraz
- Función heurística de búsqueda más corta a la solución
- Se analizan todas las posibilidades de los sucesores con similares características para elegir la mejor y expandirla.

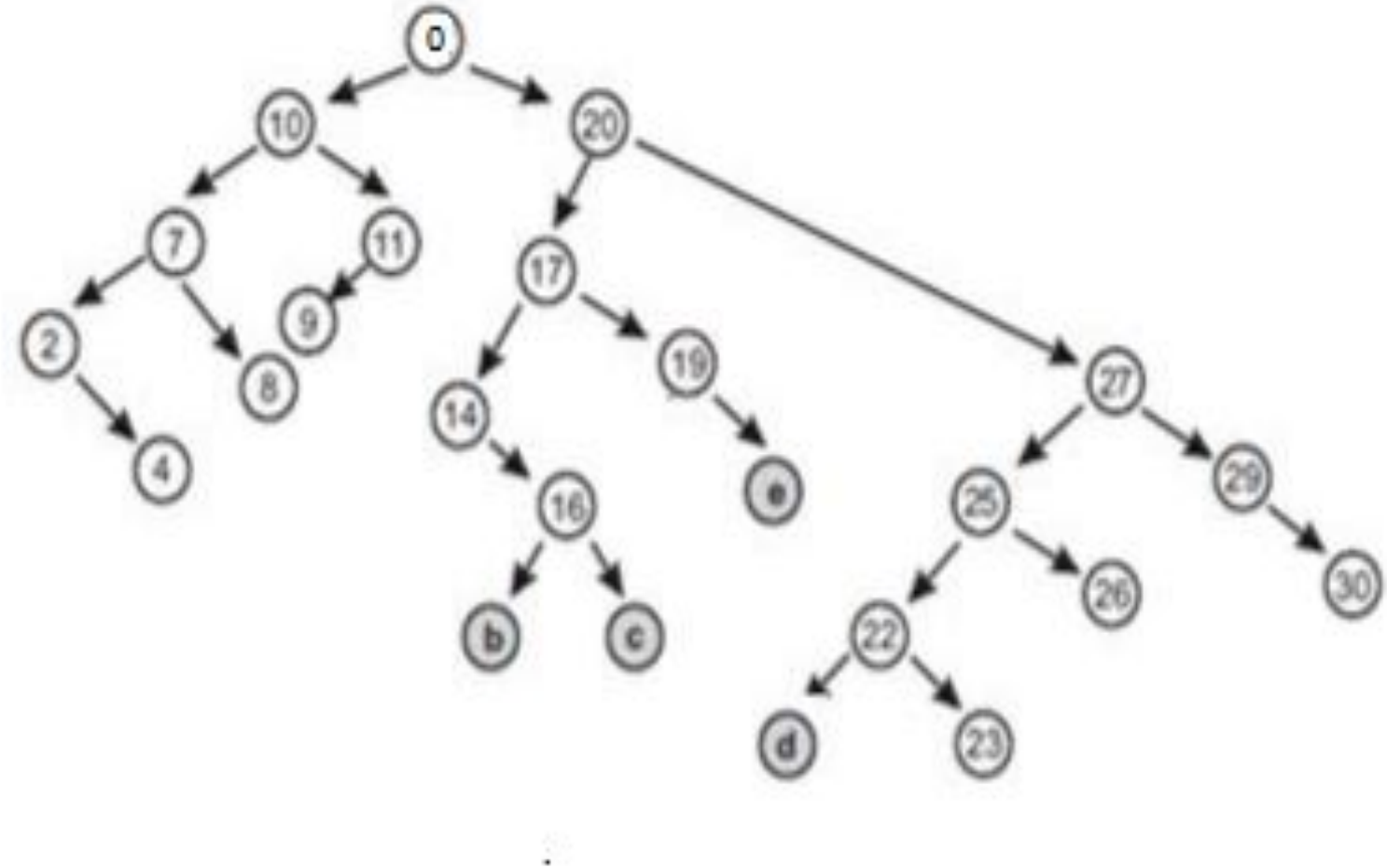


Búsqueda escalada simple y primero el mejor(cont.)

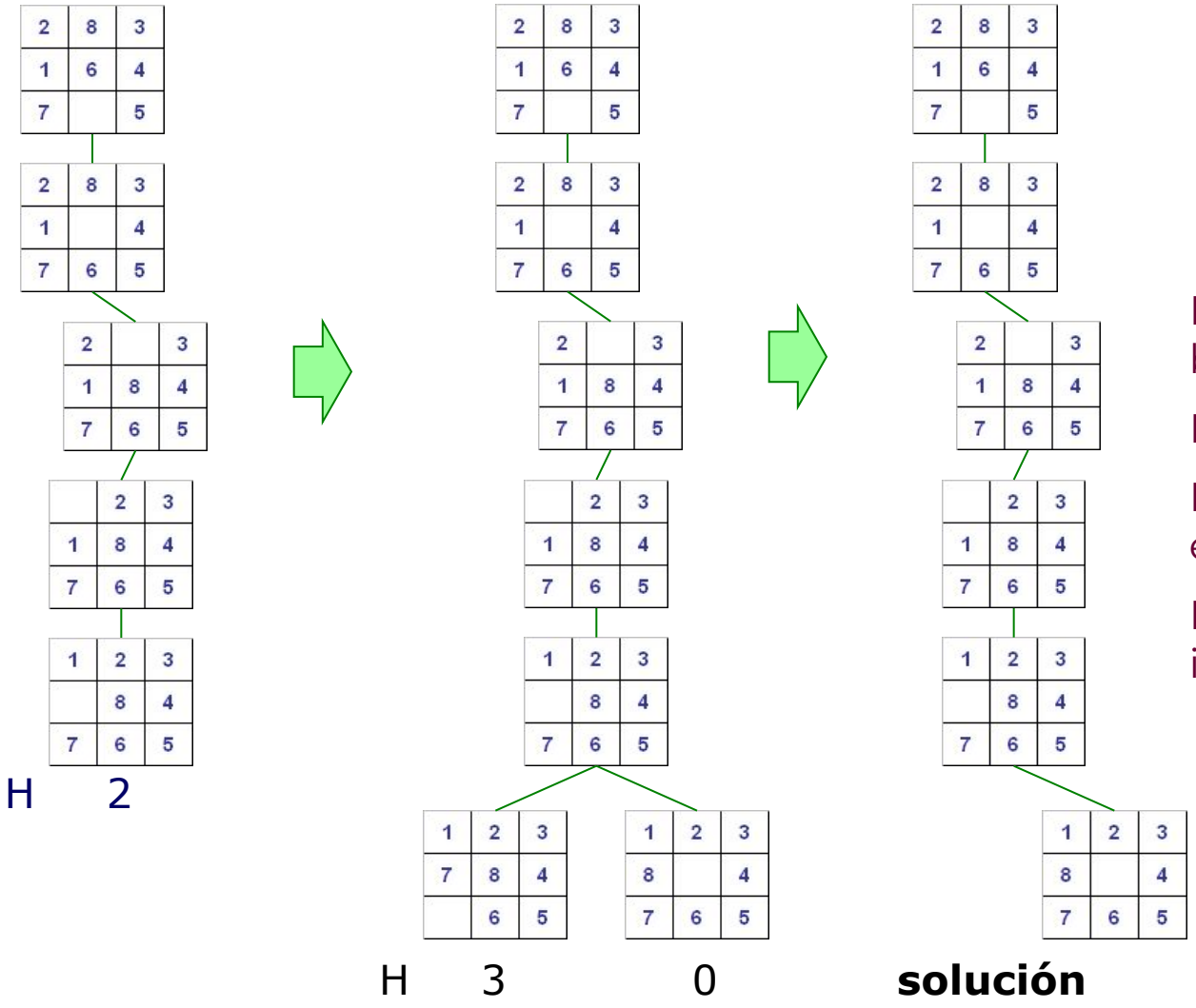
Ejemplo del rompecabezas numérico



Búsqueda escalada escala simple y primero el mejor (cont.)



Búsqueda escalada simple y primero el mejor (cont.)



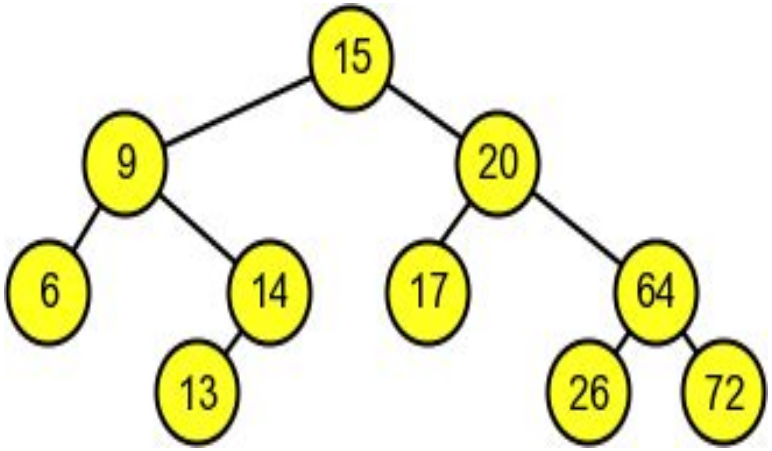
La solución para búsqueda:

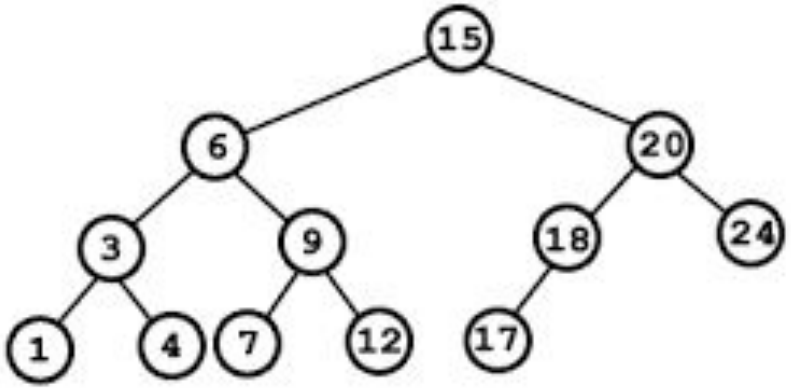
En escalada 10 estados.

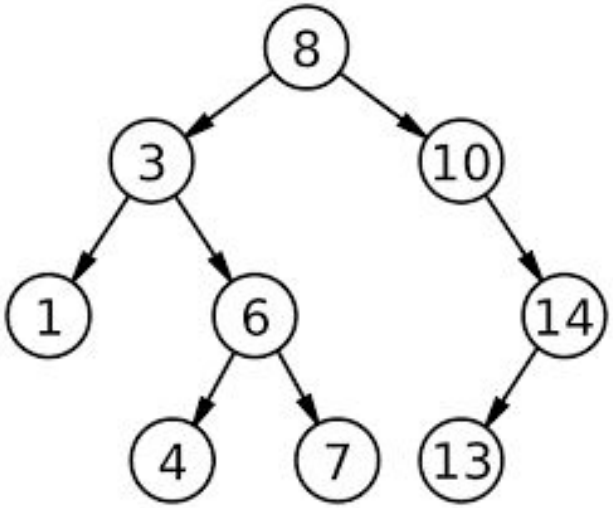
El primero mejor 6 estados

Incluyendo el estado inicial y objetivo.

Búsqueda escalada simple y primero el mejor (cont.)







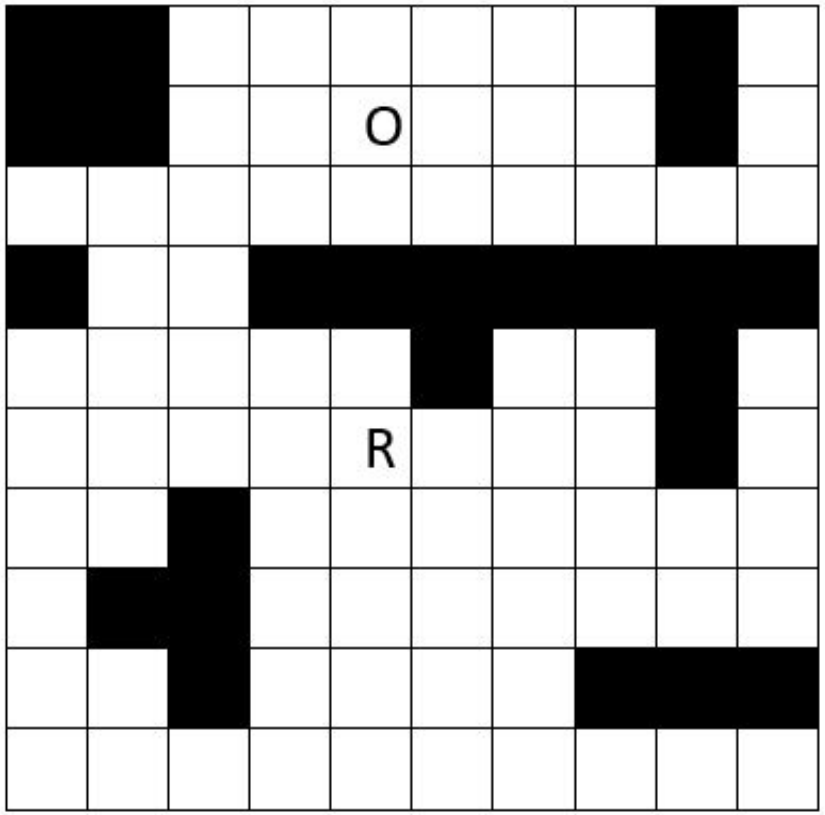
Ejercicios a resolver
Generar los diagramas de flujo de las funciones que resuelven la búsqueda en los diagramas propuestos.

2.3.3 Algoritmo A*

- Este algoritmo puede funcionar tanto sin costos de movimiento, como con costo de movimiento. En el caso de que se utilicen los costos en el movimiento, la ruta, en lugar de ser las más costosas en distancia, será la más barata en costos. También nos permite que coloquemos nuestra propia forma de llevar a cabo el cálculo del movimiento.

- Podemos utilizar el algoritmo de relleno por inundación para crear nodos equidistantes y de esta forma pensaríamos que cada nodo se encuentra en el centro de la celda. La otra opción es crear un grafo de la forma tradicional en un mapa de navegación y usar el algoritmo con los grafos existentes.

Continuación



El universo propuesto es sencillo: tiene 10 columnas y 10 renglones. Las celdas blancas son espacios vacíos en los cuales puede estar el robot o el objeto por buscar. Las celdas negras no pueden utilizarse y representan obstáculos o muros, por lo que no es posible ocuparlas.

Dentro de nuestro mundo, colocamos en una celda el robot, y en otra celda, el objeto que se desea encontrar.

Coloquemos nuestro robot en una celda, y el objeto, en otra.

Elementos del algortimo

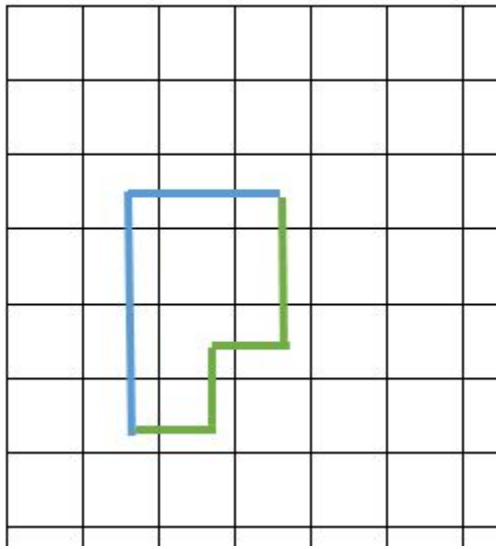
- Necesitamos dos listas: lista abierta y cerrada.

- En la lista abierta, guardamos las celdas que tenemos que revisar dentro del algoritmo. Conforme vayamos avanzando, se agregarán nuevas celdas a esta lista, pero al principio sólo contendrá la celda donde está el robot. En la lista cerrada, vamos a colocar las celdas que han sido visitadas y calculadas. Las celdas pasarán de la lista abierta a la lista cerrada, conforme se avance en el algoritmo.

Continuación …

- Para cada celda que se procesa en el algoritmo, necesitamos una variable; ésta guarda el valor del puntaje. El puntaje nos permite seleccionar la mejor ruta. Para calcularlo, necesitamos dos valores que se suman. El primer valor es el costo desde el inicio. Este indica la cantidad de movimientos necesarios para llegar desde esa celda hasta el robot. Cada movimiento nos permite avanzar solamente una celda. El segundo valor necesario para el puntaje se conoce como heurística; podemos colocar una fórmula diseñada por nosotros de acuerdo con las necesidades de nuestra aplicación. En este ejemplo utilizaremos como heurística la cantidad de movimientos que debemos hacer para llegar de la celda que se está procesando al objeto, sin que importen los obstáculos.

Continuación ...



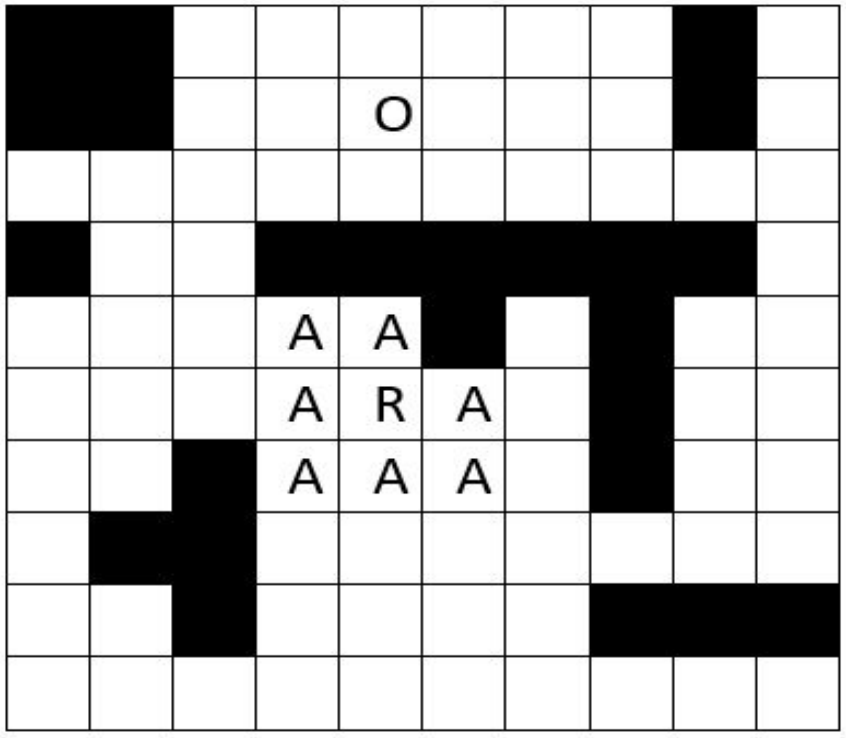
En la figura observamos 2 rutas. Para ambas, la distancia es de 5. Esto se debe a que, en ambas, la suma de movimientos horizontales y verticales es de cinco. Como podemos ver. No es posible viajar de forma transversal en este tipo de geometría.

El último dato que tenemos que tener en cada celda es la dirección hacia su padre. Gráficamente la vamos a representar por medio de referencia o un apuntador, dependiendo de cómo queramos que se implemente. Por este medio podemos trazar la ruta que nos interesa obtener.

Continuación …

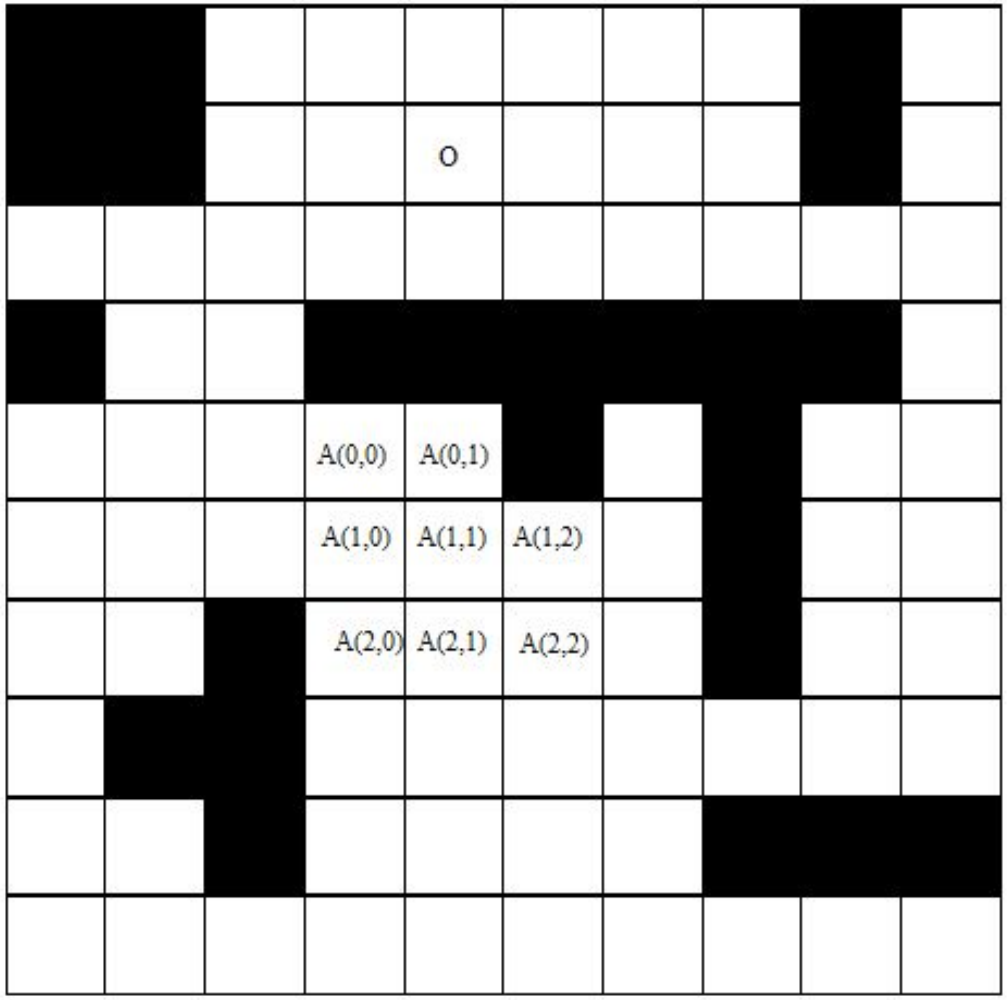
- Para arrancar el algoritmo, necesitamos colocar la celda donde se encuentra el robot en la lista abierta. El algoritmo se repetirá mientras la lista abierta tenga celdas y la celda actual no sea igual a la celda objetivo, es decir, la celda donde se encuentra el objeto al que queremos llegar; y obtener la ruta a partir de la información calculada y recolectada.

- El primer paso consiste en tomar la celda con menor puntaje, en este caso es la celda del robot, ya que es la única que existe en la lista en este momento. Después de tomar la celda, esta celda es la actual y la movemos de la lista abierta a la lista cerrada.



El valor de la heurística va a variar entre las celdas, ya que tenemos que contar la cantidad de movimientos necesarios desde la celda que estamos calculando hasta la celda donde está el objeto. Recordemos que no tomamos en cuenta los obstáculos.

El cálculo del puntaje se lleva a cabo sumando la heurística y la distancia; las celdas tendrán diferentes puntajes. No olvidemos indicar el padre de la celda que se está calculando en este caso, el padre para todas las celdas es la celda del robot.

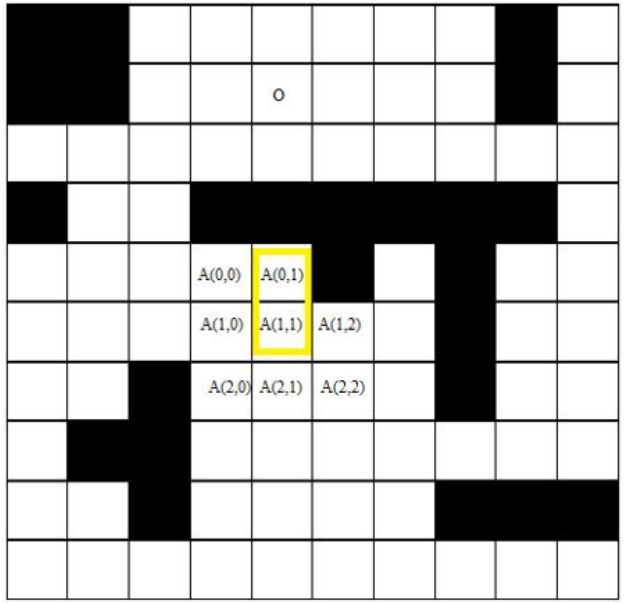


A=
Abierto
A(0,0)
D=1
H=3
P=4
A(0,1)
D= 1
H=3
P=4
A(1,0)
D=1
H=4
P=5
A(1,2)
D=1
H=4
P=5

A(2,0)
D=1
H=5
P=6
A(2,1)
D=1
H=4
P=5
A(2,2)
D=1
H=5
P=6

A(1,1)
Robot

Continuación

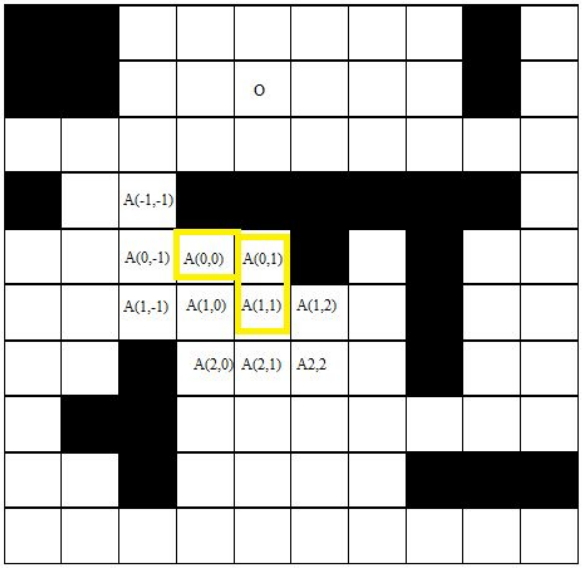


A= Abierto

A(0,0)	A(1,0)	A(2,1)	A=(1,2)
D=1	D= 1	D=1	D=1
H=3	H=4	H=4	H=4
P=4	P=5	P=5	P=5
A(0,1)	A(2,0)	A(2,2)	
D= 1	D=1	D=1	
H=3	H=5	H=5	
P=4	P=6	P=6	

El ciclo se repite, ya que aún no llegamos a los objetivos y nuestra lista abierta no está vacía. Debemos seleccionar la celda con el menor puntaje y retirar el proceso. Si observamos cómo se encuentran nuestras celdas, vemos dos celdas que tienen el puntaje 4. Cuando tenemos más de una celda, simplemente, seleccionamos la primera que esté en la lista y solucionamos el problema. Tomemos la celda norte y observemos qué sucede.

La celda se manda a la lista cerrada. Luego observamos sus vecinos, pero ninguna de las celdas vecinas se puede adicionar a la lista abierta. Esto se debe a que son obstáculos, ya están en la lista abierta o se encuentran en la lista cerrada. Por lo tanto, ninguna celda se adiciona y repetimos nuevamente el ciclo



Continuación ...

- Nuevamente reiteramos el ciclo. En ese caso, el menor puntaje de una celda es 4, y la convertimos en nuestra celda actual. Se manda a la lista cerrada y, al observar las celdas vecinas, verificamos que es posible adicionar tres nuevas celdas a la lista abierta, por lo que es necesario calcular los valores correspondientes para cada una de ellas.

Lista abierta

A= Abierto
A(0,0)
D=1
H=3
P=4
A(1,0)
D=1
H=4
P=5
A(1,-1)
D=2
H=4
P=6
A(1,2)
D=1
H=4
P=5
A(2,0)
D=1
H=5
P=6
A(2,1)
D=1
H=5
P=6
A(2,2)
D=1
H=5
P=6
A(-1,-1)
D=2
H=2
P=4
A(0,-1)
D=2
H=3
P=5

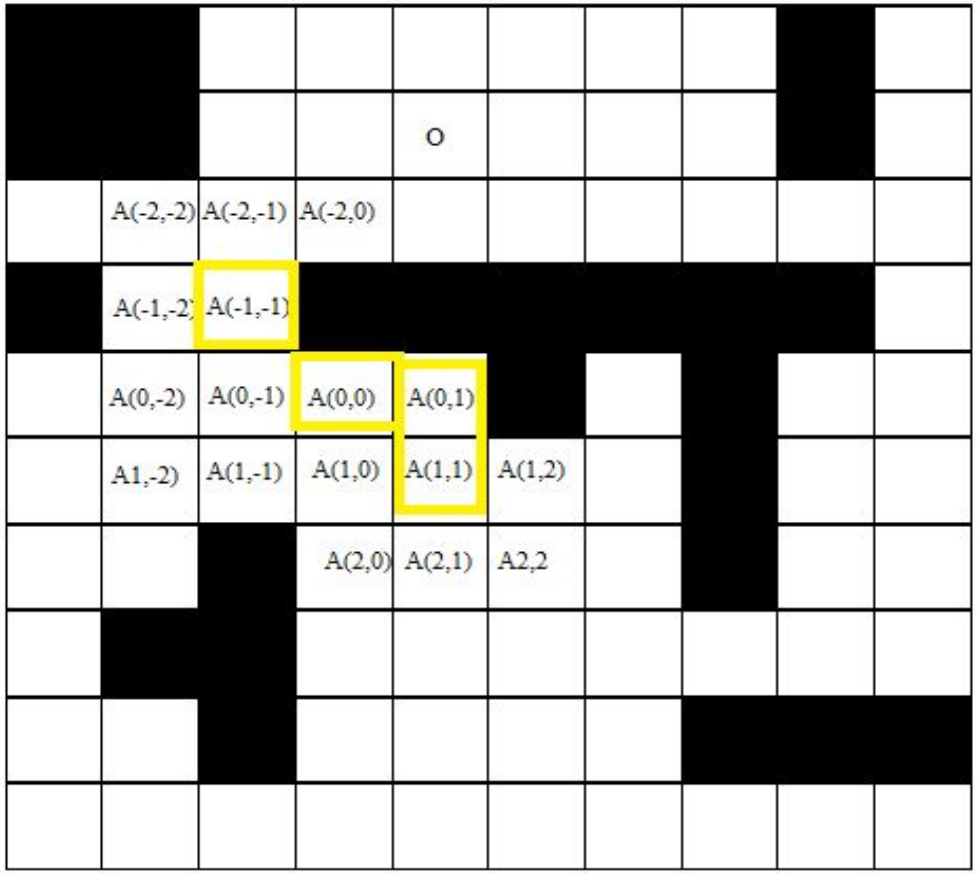
Lista Cerrada

A(0,1)
D= 1
H=3
P=4

Continuación …

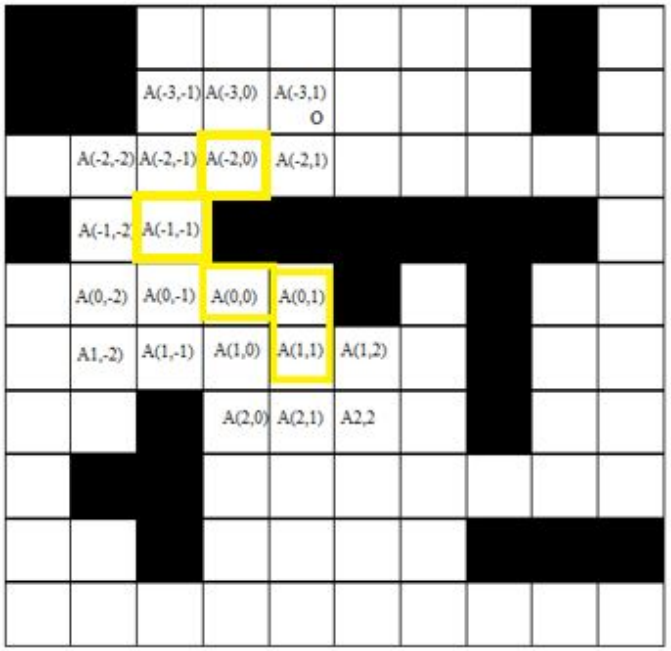
- Nuevamente, verificamos nuestra lista abierta. En este caso, volvemos a encontrar que la celda con el menor puntaje es la que tiene 4. La seleccionamos y la mandamos a la lista cerrada. Al revisar sus celdas vecinas, encontramos que podemos adicionar cinco celdas a la lista abierta: una celda es obstáculo, otra celda ya está cerrada y la última celda se encuentra en la lista abierta.

- Procedemos entonces a calcular los puntajes de las nuevas adiciones. Podemos observar que nuestra lista abierta ha crecido conforme avanzamos por el algoritmo: en este momento, tenemos doce celdas en ella, sin embargo muchas de esas celdas no nos llevarán a una ruta optima y, después de haber procesado cuatro celdas, nos hemos acercado bastante.



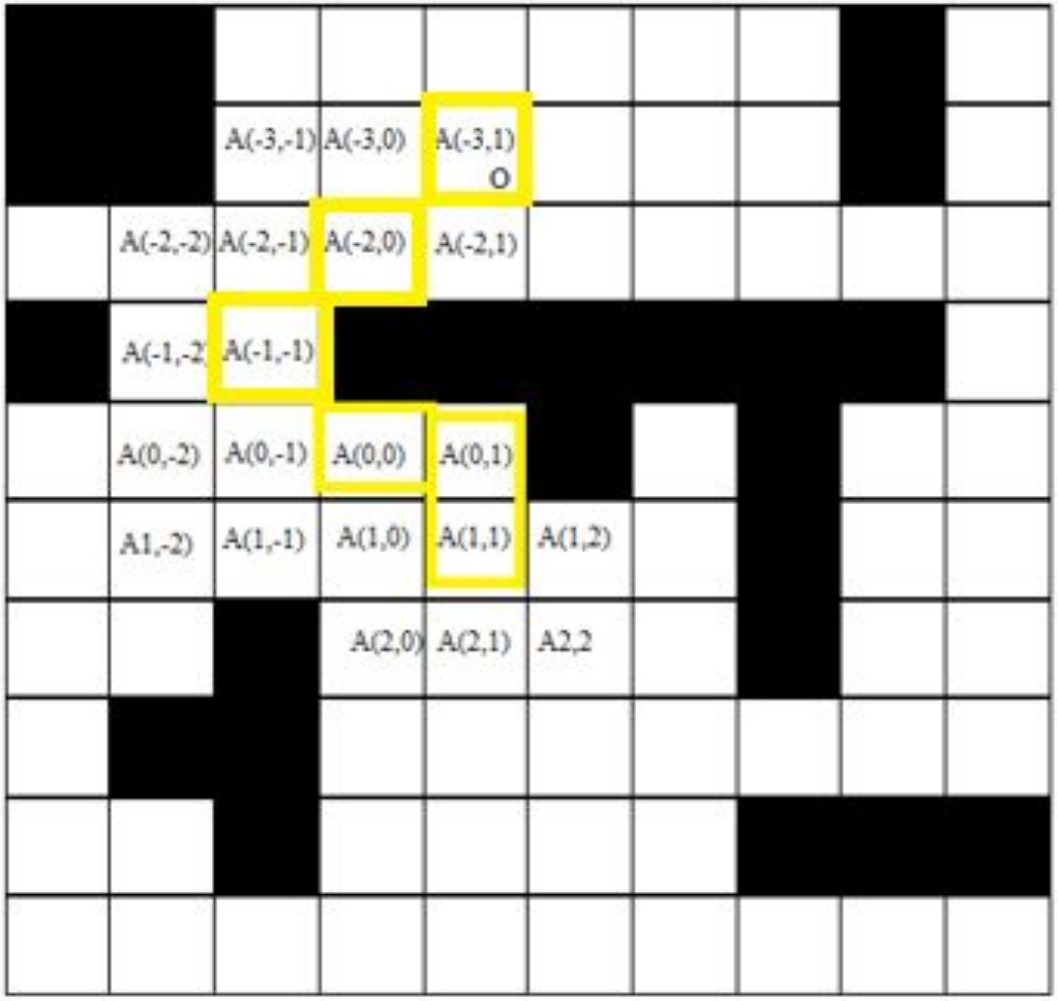
Ejercicio
Determinar los valores de la lista abierta y cerrada que permiten elegir la posición A(-1, -1)

Continuación ...



- Para nuestra celda actual podemos adicionar cuatro nuevas celdas a lista abierta, incluso una de esas celdas es la del objeto; pero el algoritmo no finalizará hasta que la celda actual sea la del objeto. Dos celdas son obstáculo, una celda ya está en la lista cerrada y otra se encuentra en la lista abierta. Como siempre, se calculan los pesos de cada celda adicionada a la lista abierta.

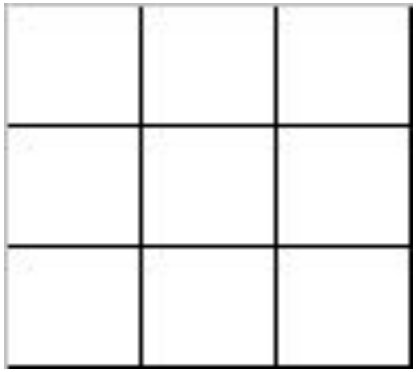
- Nuevamente corremos el ciclo del algoritmo y seleccionamos la celda con el puntaje más bajo de la lista abierta. La celda seleccionada en este caso es nuestra celda objetivo, por lo que podemos salir del algoritmo sin necesidad de procesar nada más, hemos finalizado el proceso.


A(-3,-1)
A(-3,0)
A(-3,1)
O
A(-2,-2)
A(-2,-1)
A(-2,0)
A(-2,1)
A(-1,-2
A(-1,-1)
A(0,-2)
A(0,-1)
A(0,0)
A(0,1)
A1,-2)
A(1,-1)
A(1,0)
A(1,1)
A(1,2)
A(2,0)
A(2,1)
A2,2

Ejercicio

Con cualquiera de los métodos vistos, resuelva el problema del cuadrado mágico.

Se tiene un cuadrado dividido en casillas internas de 3X3, la solución es introducir en cada casilla un número natural del 1 al 9, sin repetir; de tal manera que la suma de cada fila, cada columna y de las diagonales tengan un resultado igual.



2.4 Búsqueda adversaria

2.4.1 Algoritmo Min Max
2.4.2 Algoritmo Poda Alpha Beta

La teoría de los juegos es una rama de la matemática con aplicaciones a la economía, sociología, biología y psicología, que analiza las interacciones entre individuos que toman decisiones en un marco de incentivos formalizados (juegos).
La teoría de juegos es una herramienta que ayuda a analizar problemas de optimización interactiva.

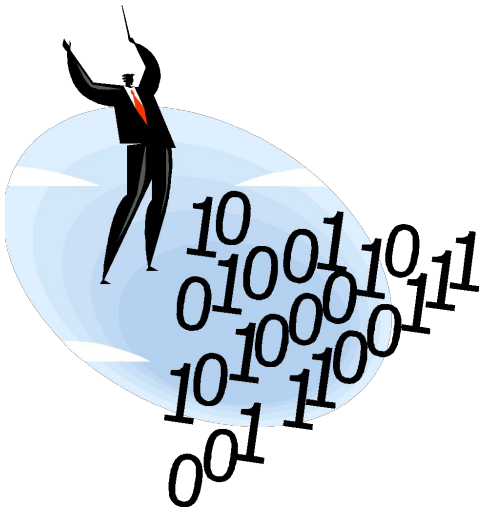
Una estrategia es un plan de acciones completo que se lleva a cabo cuando se juega el juego.
Cuando un jugador tiene en cuenta las reacciones de otros jugadores para realizar su elección, se dice que el jugador tiene una estrategia.

2.4.1 Algoritmo minimax

- Minimax es un procedimiento de Búsqueda primero en profundidad, siendo ésta la estrategia principal para los árboles de juego.

- Minimax busca para abajo hasta cierta profundidad y trata a los nodos de dicho nivel de profundidad como si fueran nodos terminales.

- Invoca una función heurística denominada función de evaluación estática, heurística o función de utilidad, para determinar los valores de esos nodos terminales.



2.4.1 Algoritmo minimax (cont.)

- ¿Completo? Sí, cuando el juego es finito (en el ajedrez hay reglas que impiden un juego infinito)
- ¿Óptimo? Sí, cuando el adversario juega perfectamente - en el otro caso Minimax es sub-óptimo
- ¿Complejidad temporal? $O(b^m)$
- ¿Complejidad espacial? O(m) - se trata de una exploración de tipo búsqueda primero en profundidad, ahorrativa de memoria.

Los problemas asociados con la técnica minimax son:

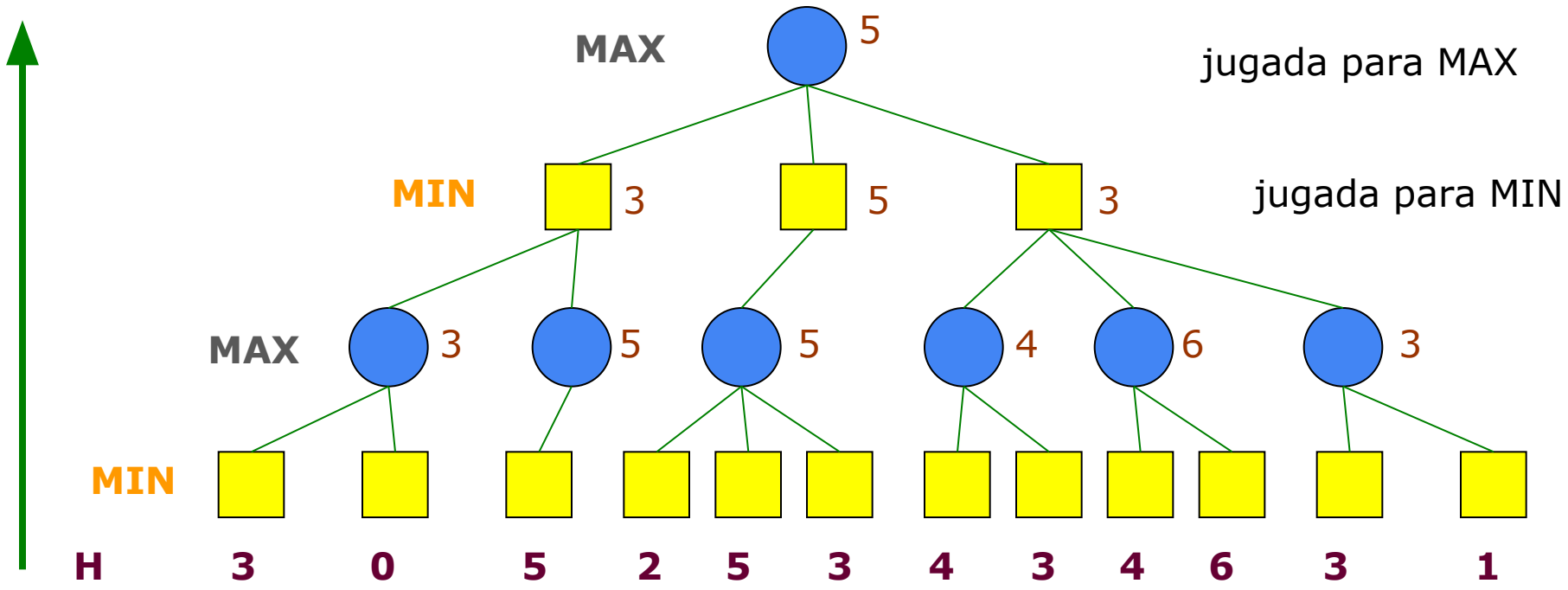
- Cuándo cortar la búsqueda para llamar a la función de evaluación estática.
- Qué función de evaluación elegir.

Interesa además

- Cuán prometedora es la ruta.
- Cuánto tiempo tiene la computadora para decidir.

2.4.1 Algoritmo minimax (cont.)

Árbol de juego general usando minimax.



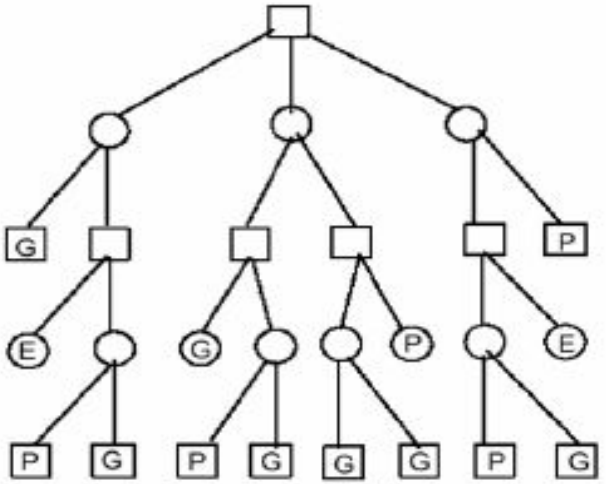
2.4.1 Algoritmo minimax (cont.)

Juego del gato.

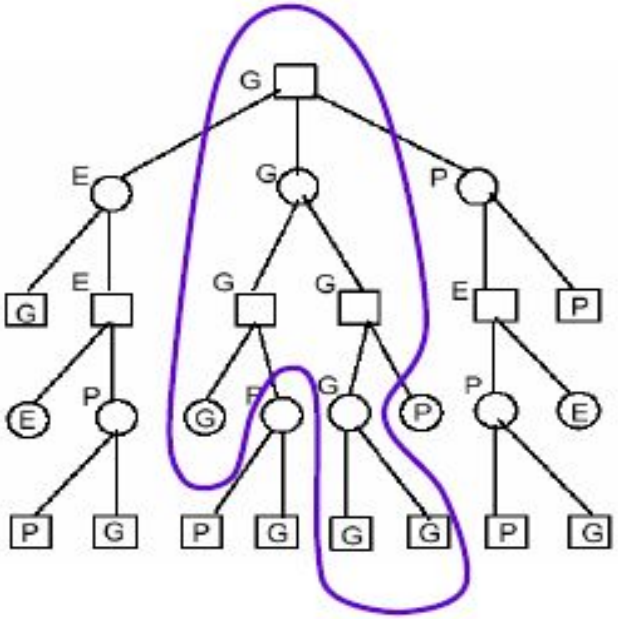
- Estados: tablero más fichas.
- Estado inicial: tablero más ficha inicial.
- Estados finales: tableros llenos o con línea ganadora.
- Movimientos: 9 movimientos máximo posibles por jugada, uno por casilla.
- Aplicación de movimientos: aplicable si la casilla no está ocupada.
- Función de utilidad o heurística:
  - 1 si es “ganador” para MAX.
  - 0 si es empate.
  - -1 si es “ganador para MIN.

Continuación ...

Árbol de juego sin resolver



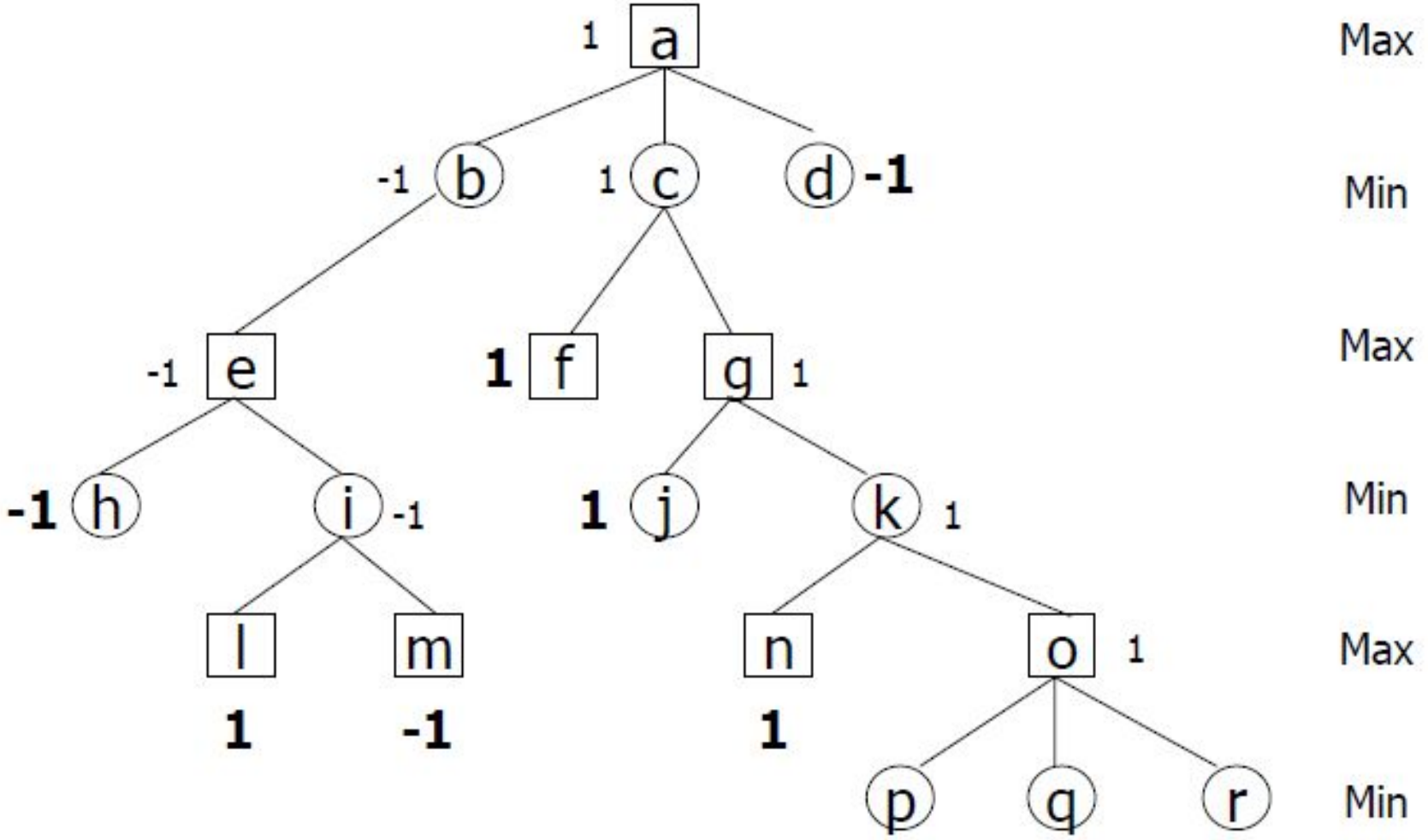
Árbol de juego resuelto



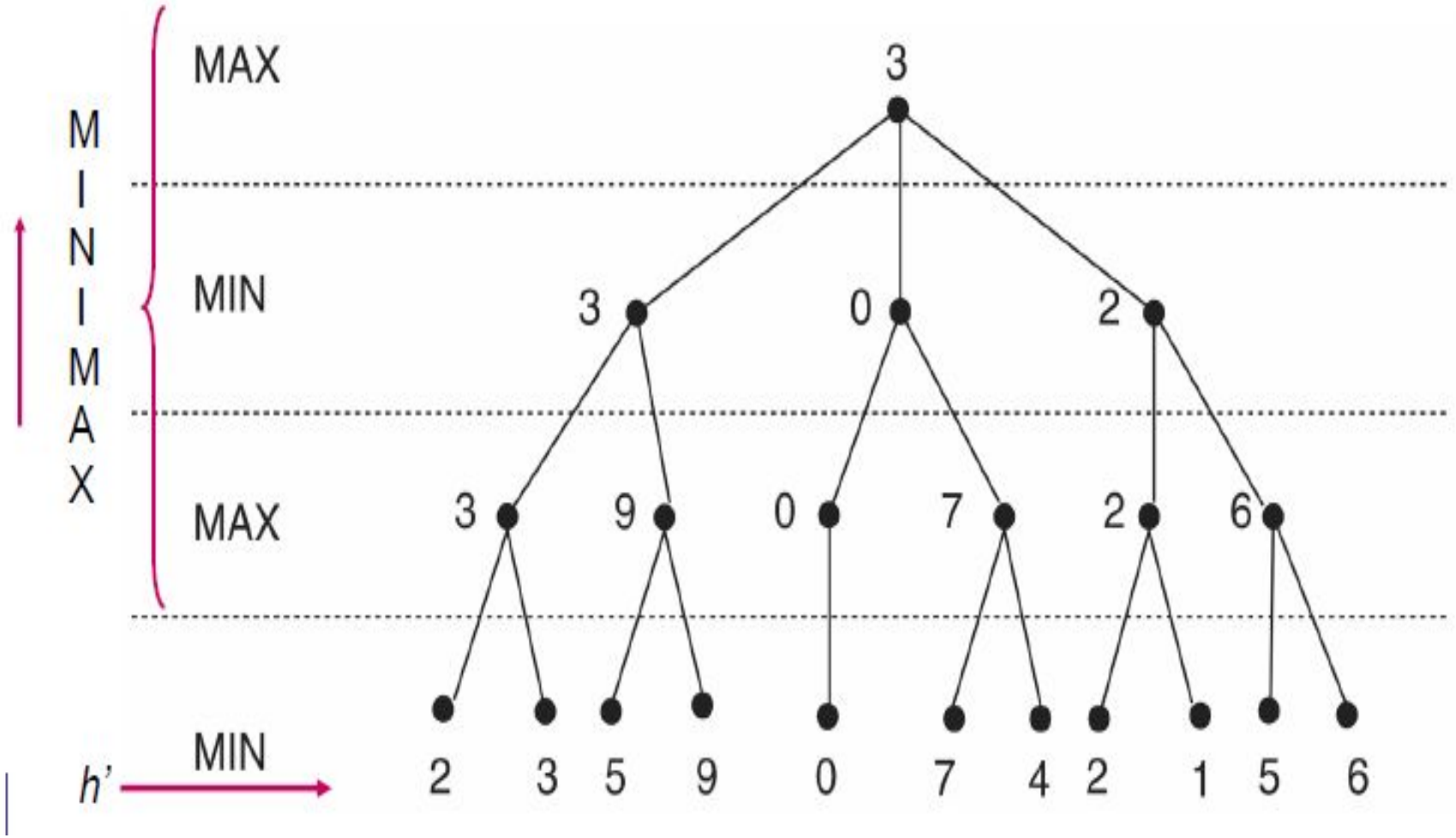
G = 1, E = 0, P = -1
Nodos MAX: cuadrados
Nodos MIN: círculos

Árbol ganador para MAX

Otro ejemplo



Ejercicio



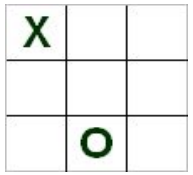
2.4.1 Algoritmo minimax (cont.)

Juego del gato (cont.)

- Función heurística: H(n)=M(n)-O(n)
- M(n) es el número de mis posibles líneas ganadoras (contando las vacías)
- O(n) es el número de sus posibles líneas ganadoras (contando las vacías).
- Si al final +: MAX gana
- Si al final - : MIN gana.

Por ejemplo: X es MAX O es MIN

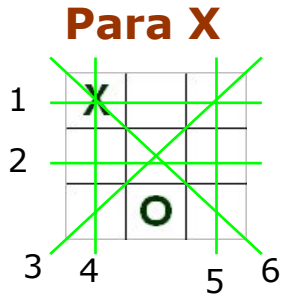
El estado

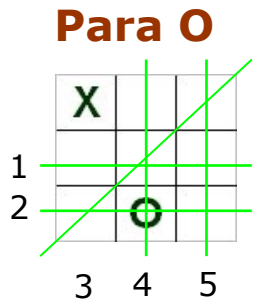


H(n)=6-5=**1**

X tiene M(n)=**6** posibles líneas ganadoras

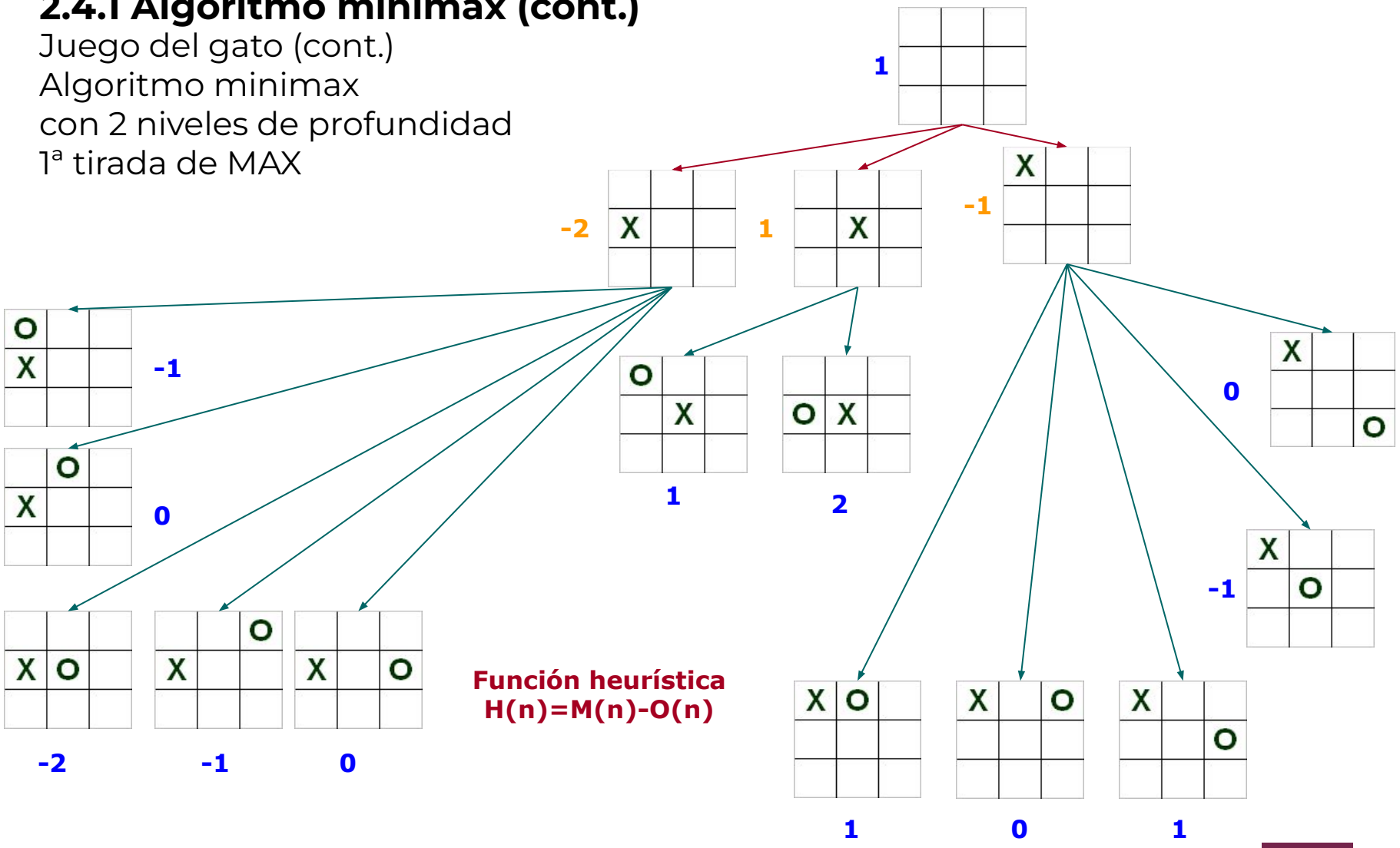
O tiene O(n)=**5** posibles líneas ganadoras





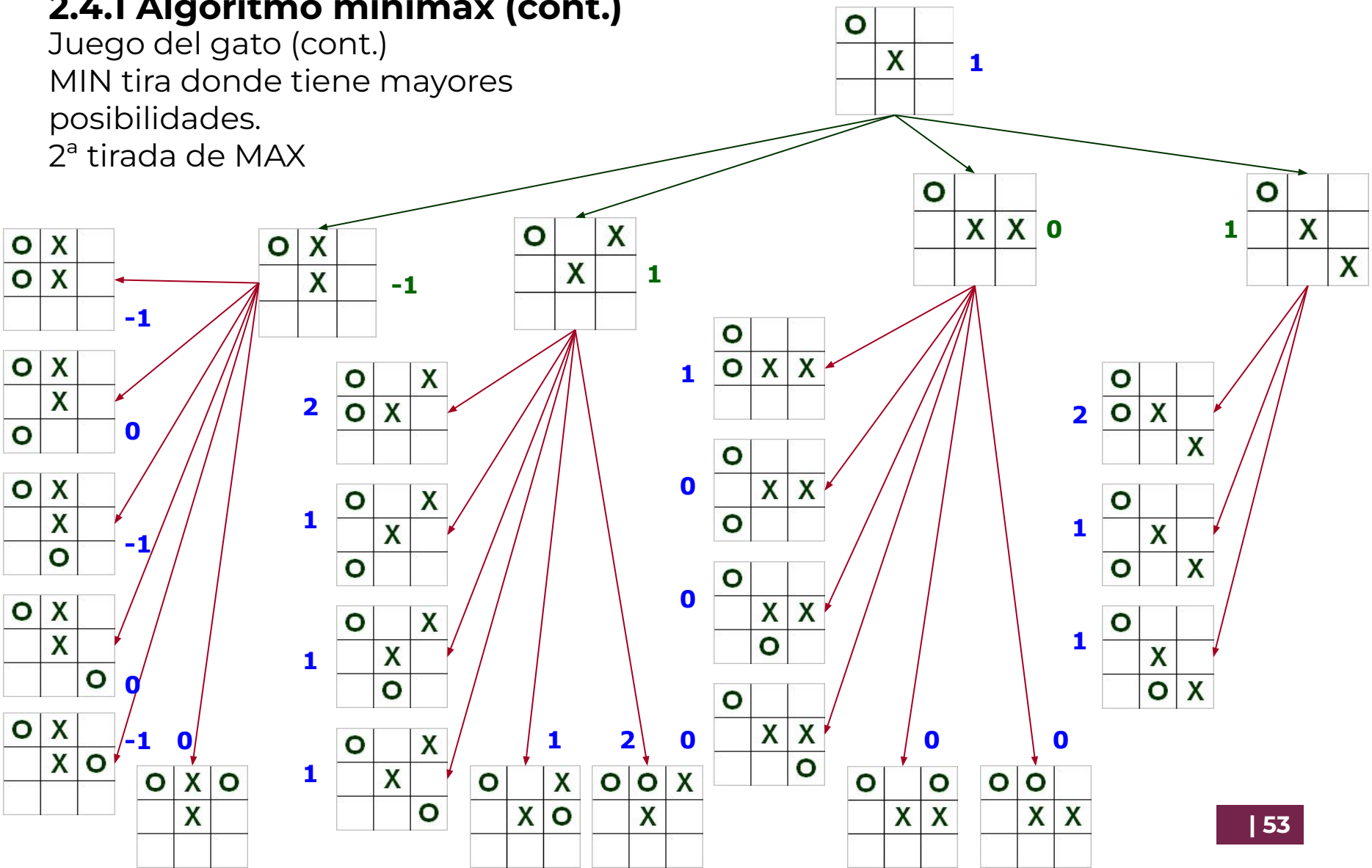
2.4.1 Algoritmo minimax (cont.)

Juego del gato (cont.)
Algoritmo minimax
con 2 niveles de profundidad
1ª tirada de MAX



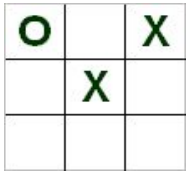
2.4.1 Algoritmo minimax (cont.)

Juego del gato (cont.)
MIN tira donde tiene mayores posibilidades.
2ª tirada de MAX



2.4.1 Algoritmo minimax (cont.)

Juego del gato (cont.)
MIN tira donde tiene mayores posibilidades.
3ª tirada de MAX



- ☺ Con estas reglas para analizar tiradas, seguramente se llega a un empate entre jugadores.
- ☺ Se crean árboles completos (primero en anchura).
- ☺ Ya creado el árbol se evalúa la función heurística.
- ☺ Para un juego más complejo es ineficiente crear árboles completos.

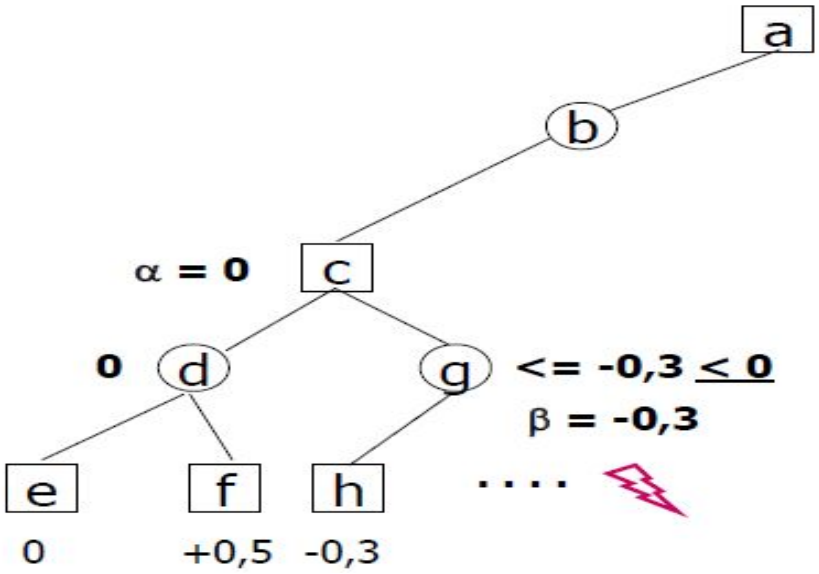
2.4.2 Poda alfa-beta

- Optimización de minimax.
- Búsqueda primero en profundidad, evaluando la heurística. Sólo se deben considerar los nodos a lo largo de un camino en el árbol.
- Para cada nodo MAX se calcula un valor alfa.
- Para cada nodo MIN se calcula un valor beta.

Continuación …

Se empieza descendiendo primero en profundidad hasta la capa límite y se calcula *h'* del estado y de sus hermanos

Si son nodos MAX, el mínimo va a al padre y este valor se ofrece al abuelo como valor provisional α
Sigue descendiendo a otros nietos, pero termina la exploración del padre si alguno de sus valores es <= α.

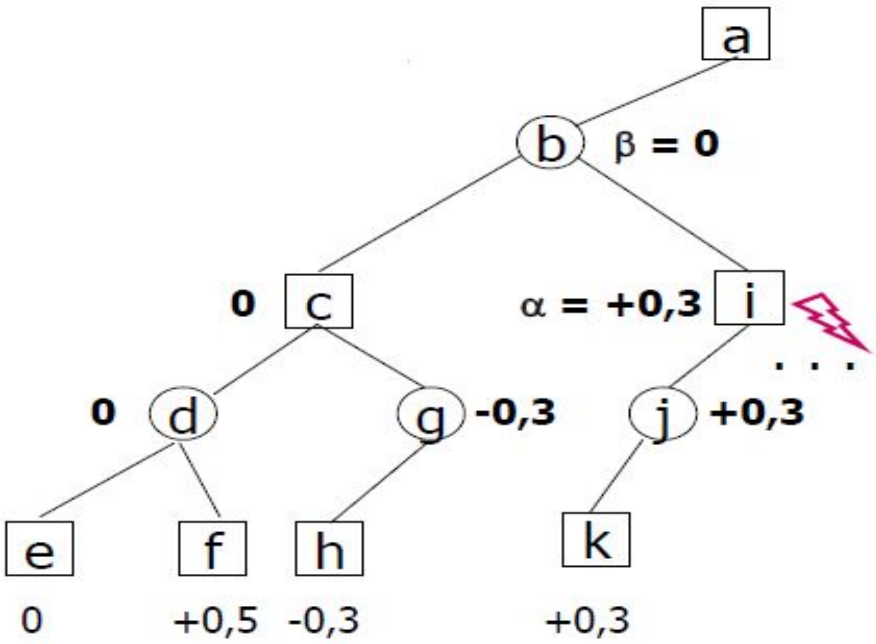


Se puede cortar la búsqueda por debajo de cualquier nodo MIN cuyo valor β sea menor o igual que el valor α de algún antecesor MAX.
A ese nodo MIN se le asigna definitivamente su valor β que puede no coincidir con el que habría obtenido minimax *(podría ser <)* pero esto no afecta a la elección de movimiento.

Continuación ...

Si son nodos MIN, el máximo va a al padre y este valor se ofrece al abuelo como valor provisional β

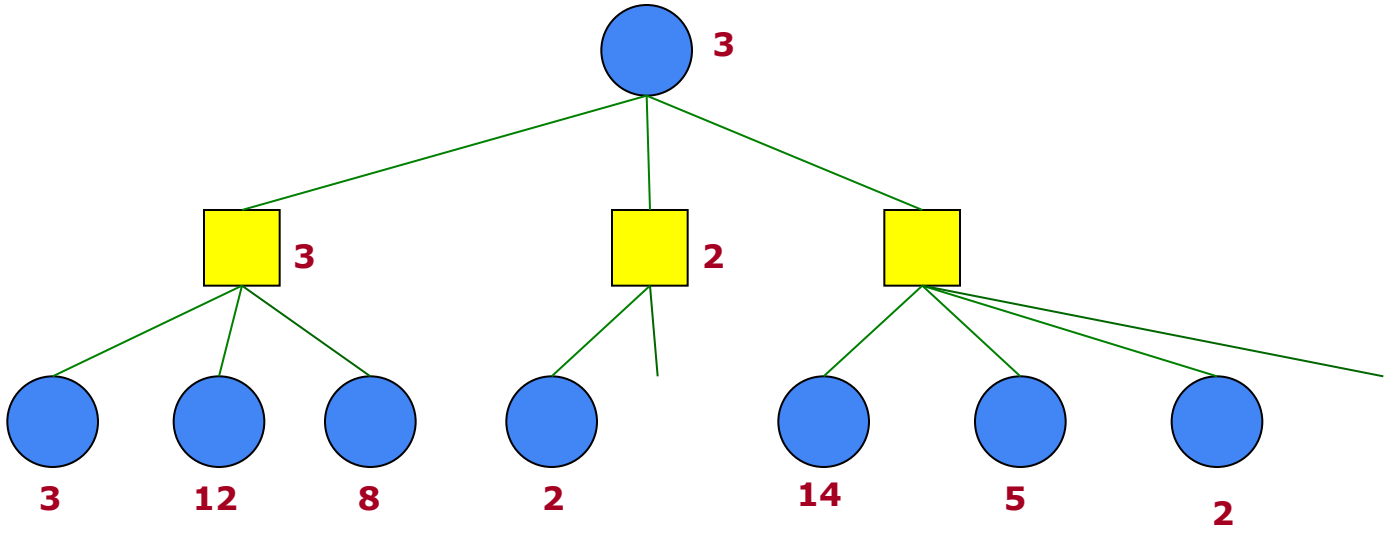
Sigue descendiendo a otros nietos, pero termina la exploración del padre si alguno de sus valores es >= β



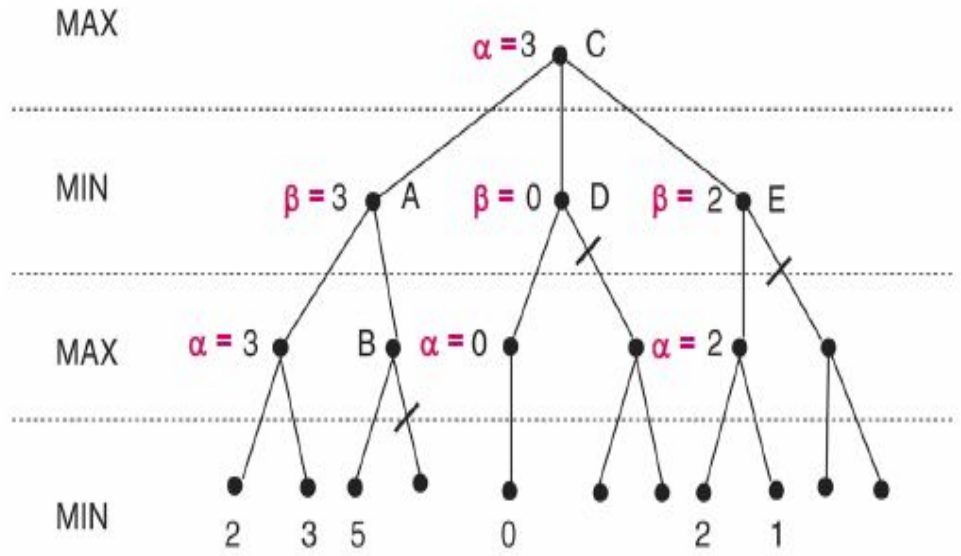
Se puede cortar la búsqueda por debajo de cualquier nodo MAX cuyo valor α sea mayor o igual que el valor β de algún antecesor MIN
A ese nodo MAX se le asigna definitivamente su valor α que puede no coincidir con el que habría obtenido minimax *(podría ser >)* pero esto no afecta a la elección de Movimiento-

Continuación …

- Optimización de minimax.
- Búsqueda primero en profundidad, evaluando la heurística. Sólo se deben considerar los nodos a lo largo de un camino en el árbol.
- Para cada nodo MAX se calcula un valor alfa.
- Para cada nodo MIN se calcula un valor beta.

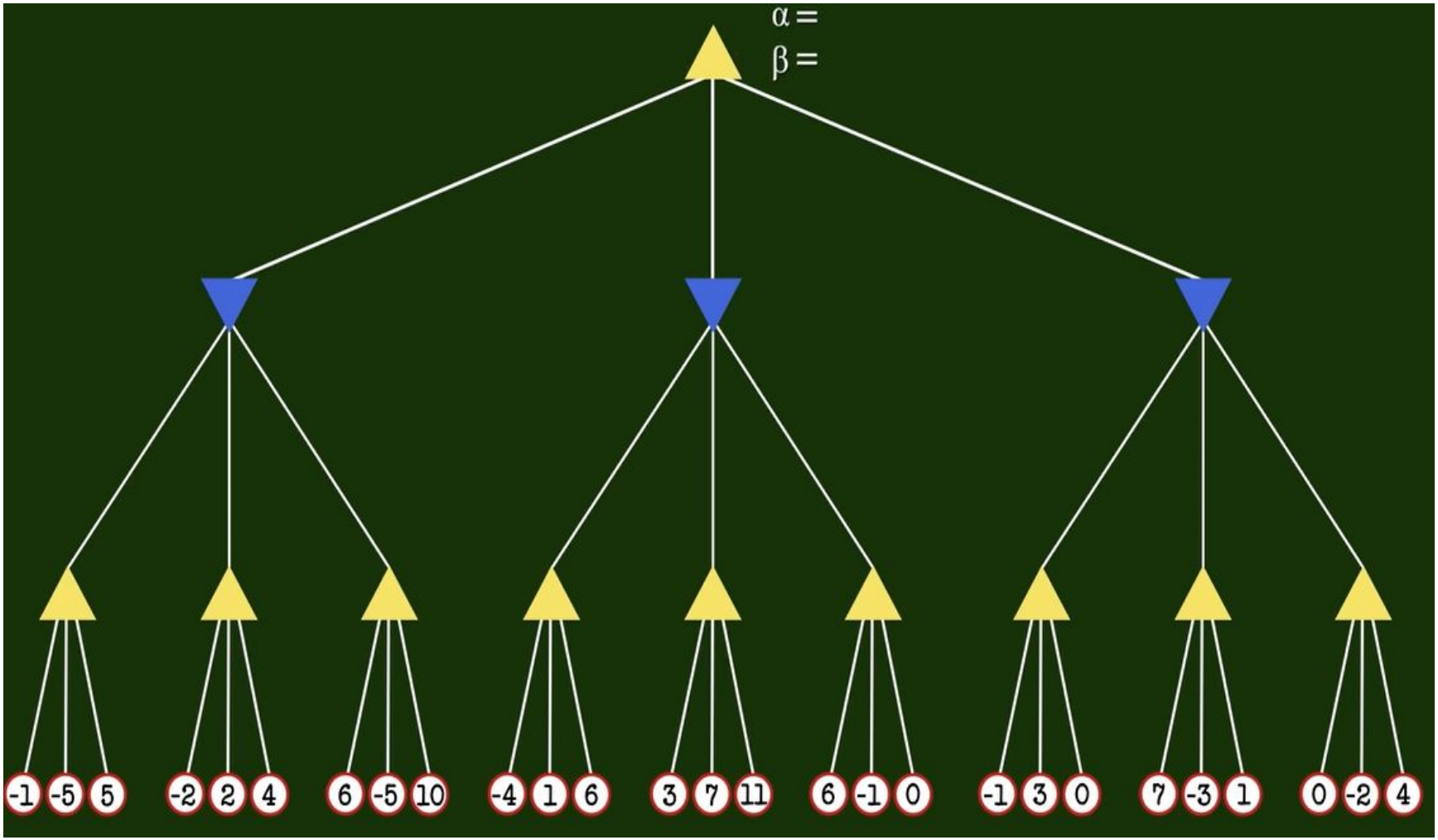


Un ejemplo



A tiene β = 3 (A <= 3)
B es β-podado (B >= 5 y 5 > 3)
C tiene α = 3 (C >= 3)
D es α-podado (D <= 0 y 0 < 3)
E es α-podado (E <= 2 y 2 < 3)
C tiene valor 3 definitivo (valor minimax)

Ejercicio



Ejercicio

Juego de Nim

- Situación inicial: pila de N fichas.
- Jugadas o reglas: tomar 1, 2, o 3 fichas de la pila por turno.
- Objetivo: obligar al adversario a tomar la última ficha.

Ejemplo de juego de Nim con 9 fichas

- Jugador 1: toma 3 fichas quedan 6
- Jugador 2: toma 1 ficha quedan 5
- Jugador 1: toma 2 fichas quedan 3
- Jugador 2: toma 2 fichas queda 1
- Jugador 1:toma 1 ficha (la última) queda 0

- Entonces el Jugador 2 gana.

Ejercicio

Elementos del juego en el NIM.

- Estados: números de fichas que quedan.
- Estado inicial: número inicial de fichas.
- Un único estado final: 0 fichas.
- El estado final es ganador para un jugador si es su turno.
- Reglas: tomar 1, 2, o 3 fichas.
- Función de utilidad heurística para el estado final:
  - +1 si le toca a MAX y
  - -1 si le toca a MIN

2.5 Comparación de algoritmos de búsqueda

- Aunque determinar cuál es el mejor algoritmo en la búsqueda depende del problema, podemos afirmar que un factor para determinar la eficiencia en la búsqueda de la solución de los niveles se ve afectado por quien genera menos estados, y llega a una solución en un tiempo menor.

- Donde la recolección de los datos es definida por los movimientos que realice cada algoritmo para completar los niveles y la cantidad de estados que genere en la búsqueda de la solución.

Coste computacional

- Depende en gran medida del número de nodos terminales examinados, es decir, del número de evaluaciones realizadas. Como vimos con el algoritmo α-β, el número de evaluaciones depende a su vez de la ordenación de los nodos terminales.

- •MiniMax: Examina todos los nodos terminales con lo cual tenemos una complejidad de Bd.

- •α-β:En el mejor de los casos la estrategia examina

- 2Bd/2-1 si d es par

- 2B(d+1)/2+B(d-1)/2-1 si d es impar

Ejercicio

- Programe diferentes algoritmos de búsqueda y determine cual es el mas eficiente.