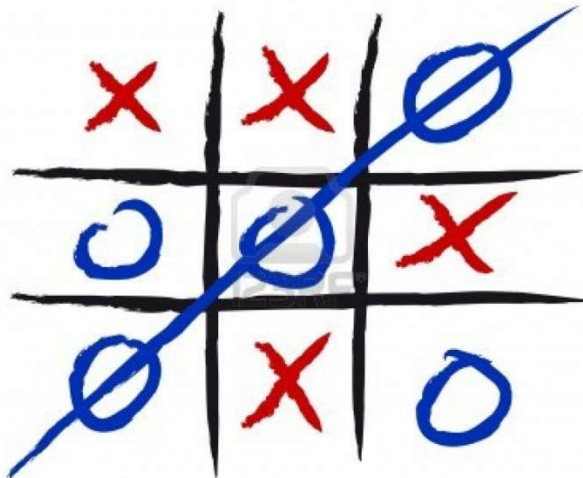


## Resolución de problemas (III.3)

### Búsqueda en juegos



# Búsqueda en juegos

---

- Tipos de juegos
- Juegos de suma cero de dos jugadores.
  - Minimax
  - Poda alfa-beta
- Juegos contra la naturaleza.
  - Repaso de probabilidades
  - Expectimax
- Juegos de suma no cero.

# Estado del arte

---

- Damas: Chinook gana el campeonato del mundo en 1994. Actualmente resuelto.
- Ajedrez: Deep Blue vence a Gary Kasparov en 1997.
- Othello (reversi): Los campeones humanos se niegan a jugar contra los ordenadores, que son mucho mejores.  
<http://es.wikipedia.org/wiki/Reversi>
- Go: AlphaGo (Google) gana al campeón del mundo (Lee Sedol) en 2016.  
<https://deepmind.com/research/alphago/>

# Tipos de juegos

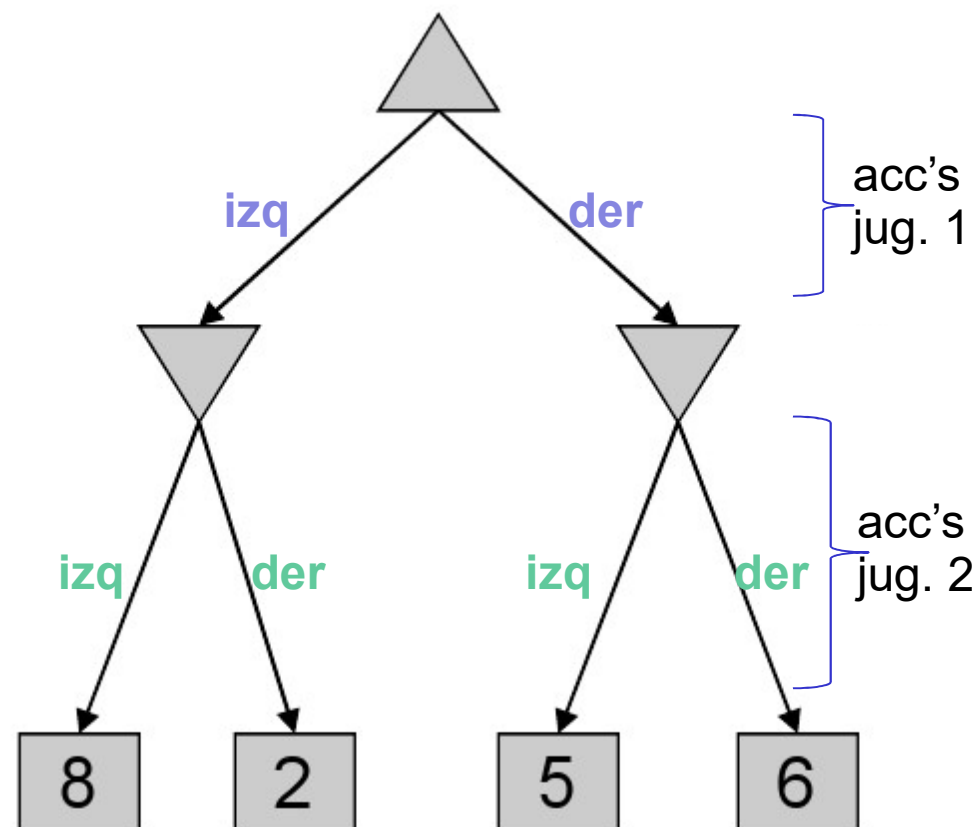
---

- Determinista vs. Estocástico.
- Uno, dos, tres, ... jugadores.
- Información completa/incompleta.
- El objetivo de los algoritmos es calcular una **política** que recomiende el movimiento a realizar en cada estado.

-

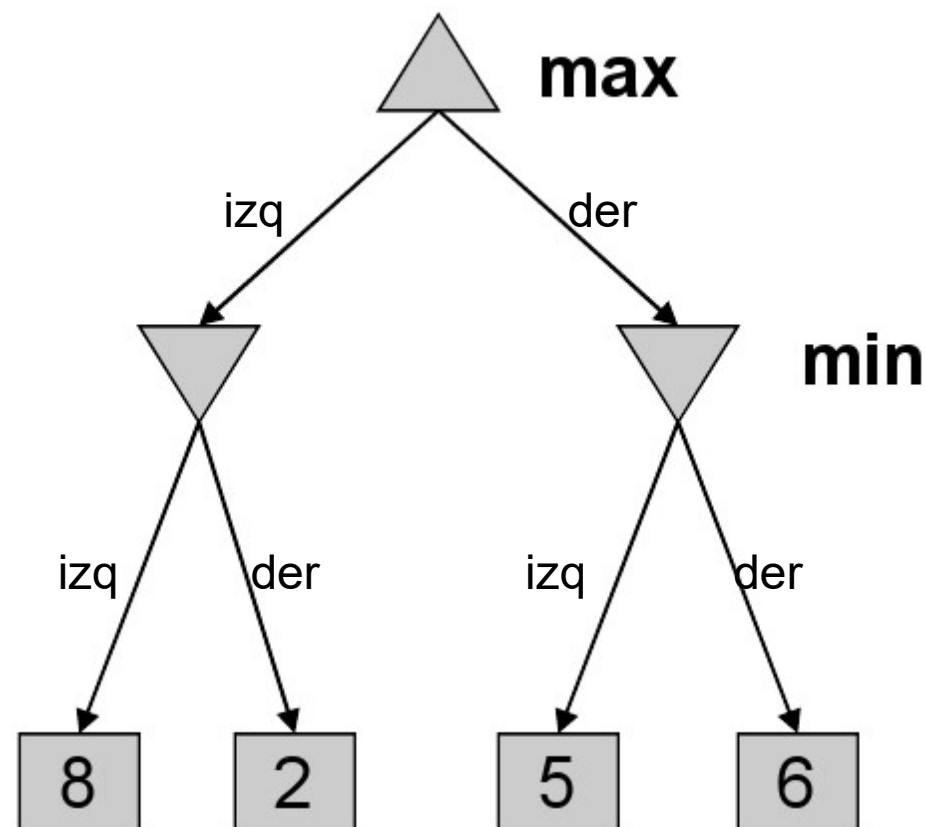
# Deterministas de dos jugadores

- Ej: Tres en raya, ajedrez, damas.
- Juegos de suma cero:
  - Lo que un jugador gana lo pierde el otro.
- ¿Qué estrategia seguir ?

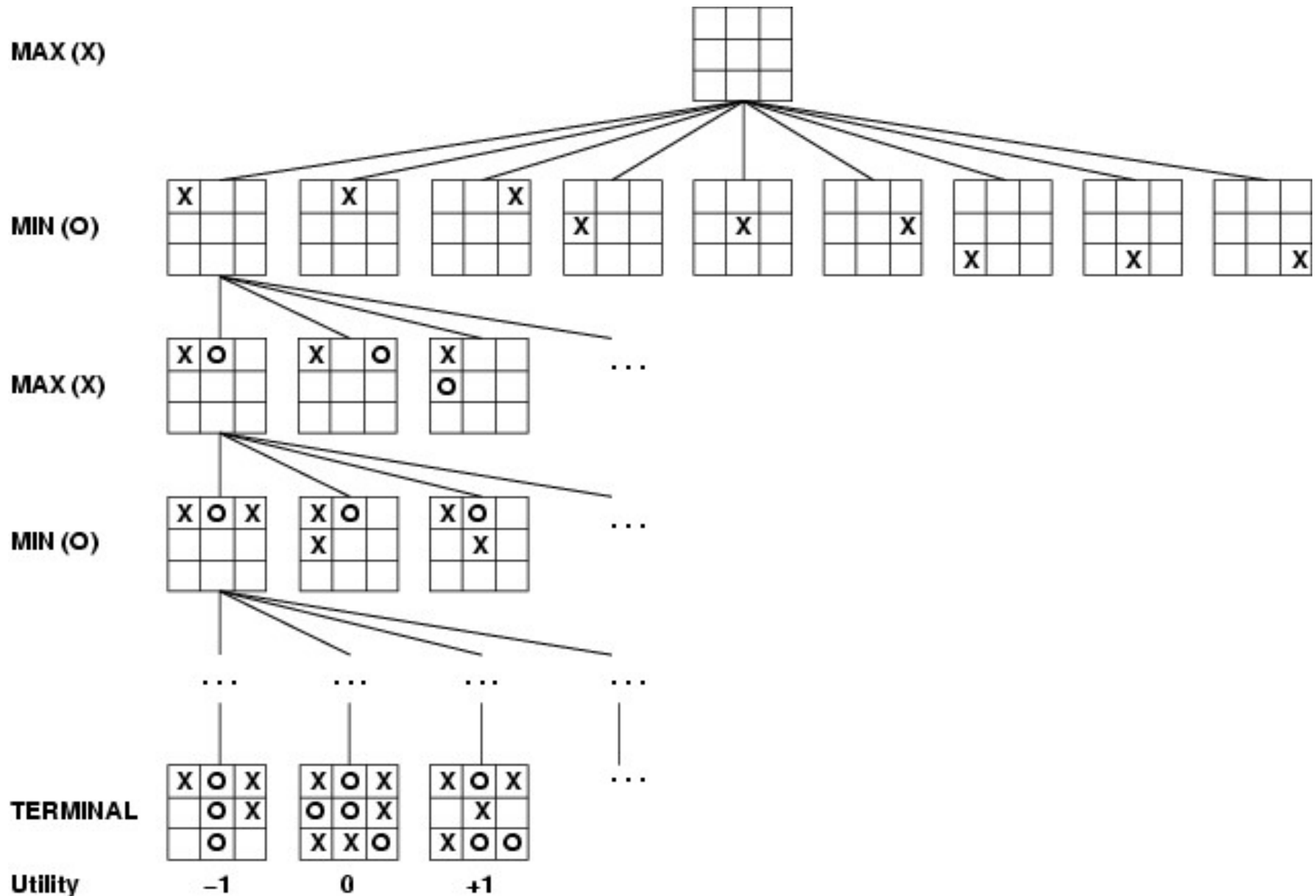


# Deterministas de dos jugadores

- Ej: Tres en raya, ajedrez, damas.
- Juegos de suma cero:
  - Lo que un jugador gana lo pierde el otro.
- Minimax:
  - Escoger el movimiento con mayor valor minimax.
  - ¿Cuál es la mejor jugada posible suponiendo que nuestro rival es perfecto?

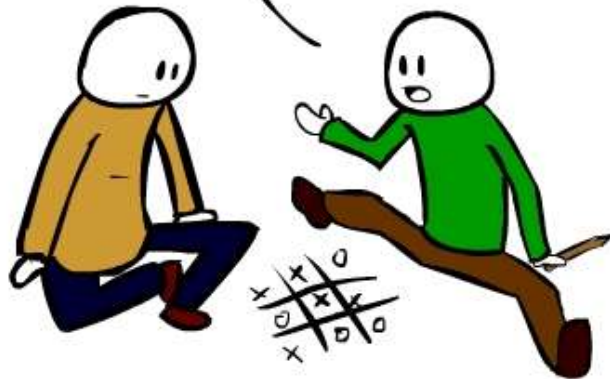


# Árbol de juego (2 jugadores, determinista, por turnos)





Esto empieza a ser aburrido



Podríamos usar otras cosas en vez de X y O



Me parece bien



Bueno, ¿qué eliges?



Me pido huecos vacíos



Yo...



Venga

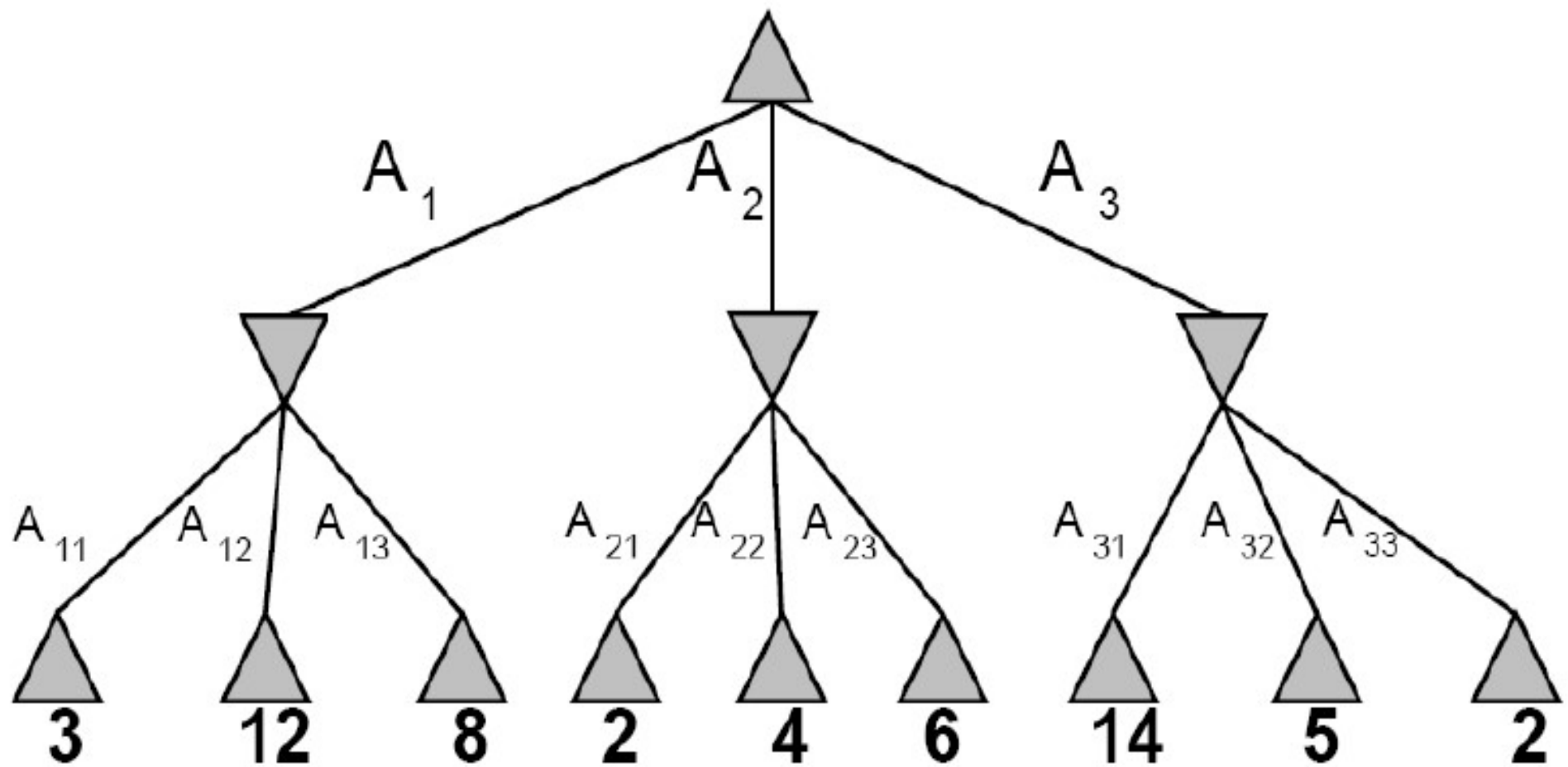


Rajado

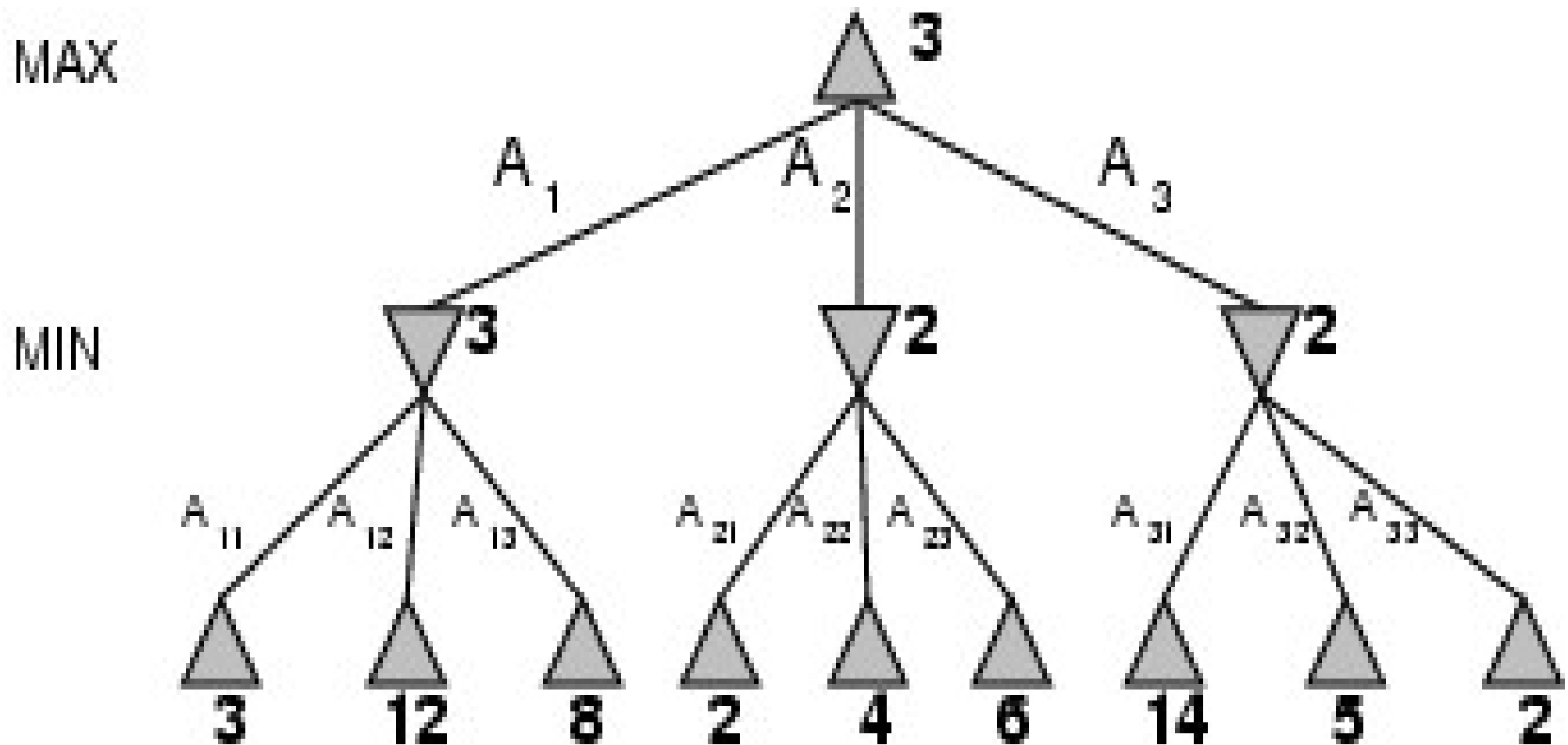


Se puede hacer peor

# Ejemplo minimax



# Ejemplo minimax



# Algoritmo minimax

**function** MINIMAX-DECISION(*state*) *returns an action*

$v \leftarrow \text{MAX-VALUE}(\textit{state})$

**return** the *action* in SUCCESSORS(*state*) with value *v*

---

**function** MAX-VALUE(*state*) *returns a utility value*

**if** TERMINAL-TEST(*state*) **then return** UTILITY(*state*)

$v \leftarrow -\infty$

**for** *a, s* in SUCCESSORS(*state*) **do**

$v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(s))$

**return** *v*

---

**function** MIN-VALUE(*state*) *returns a utility value*

**if** TERMINAL-TEST(*state*) **then return** UTILITY(*state*)

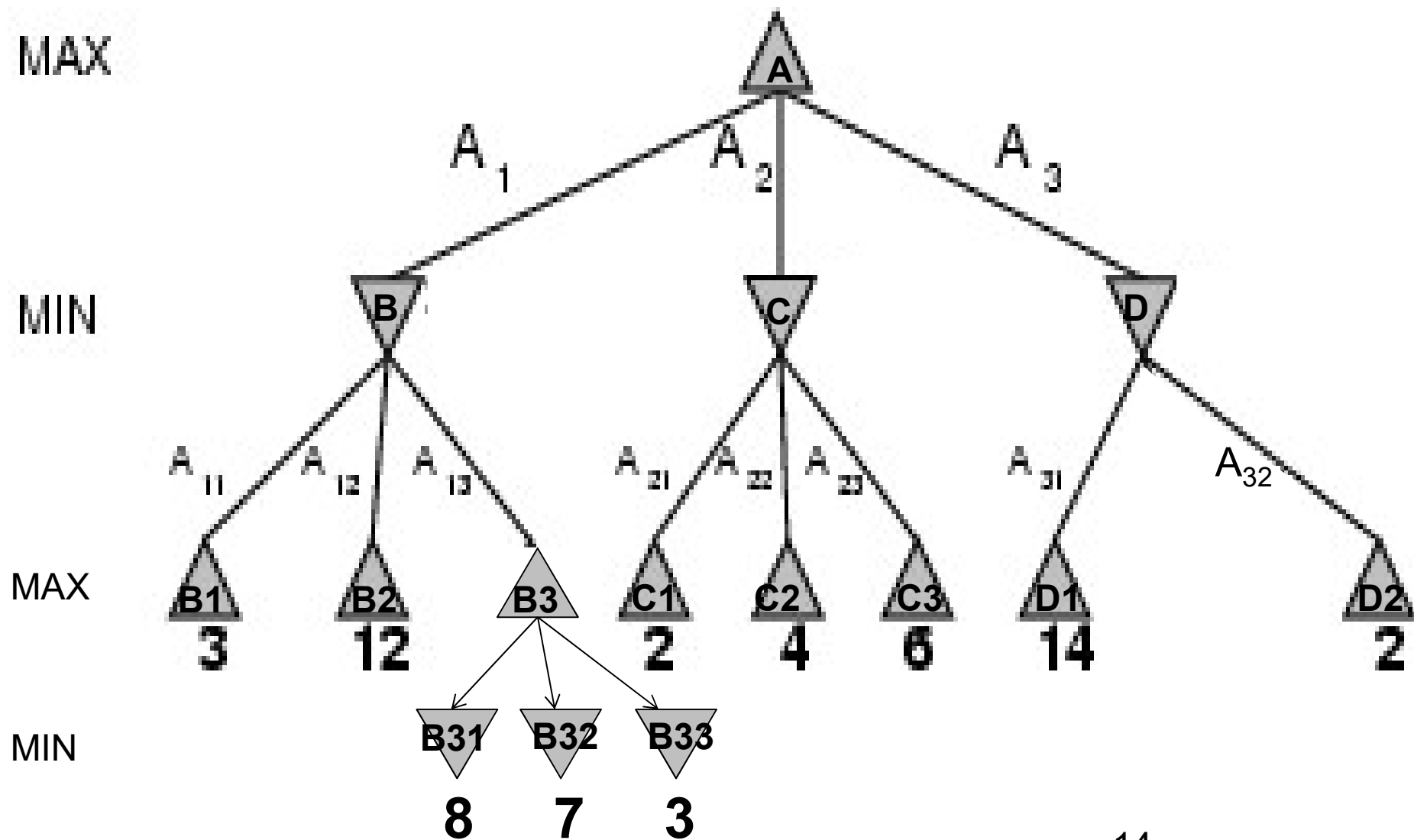
$v \leftarrow \infty$

**for** *a, s* in SUCCESSORS(*state*) **do**

$v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(s))$

**return** *v*

# Ejemplo 2 minimax



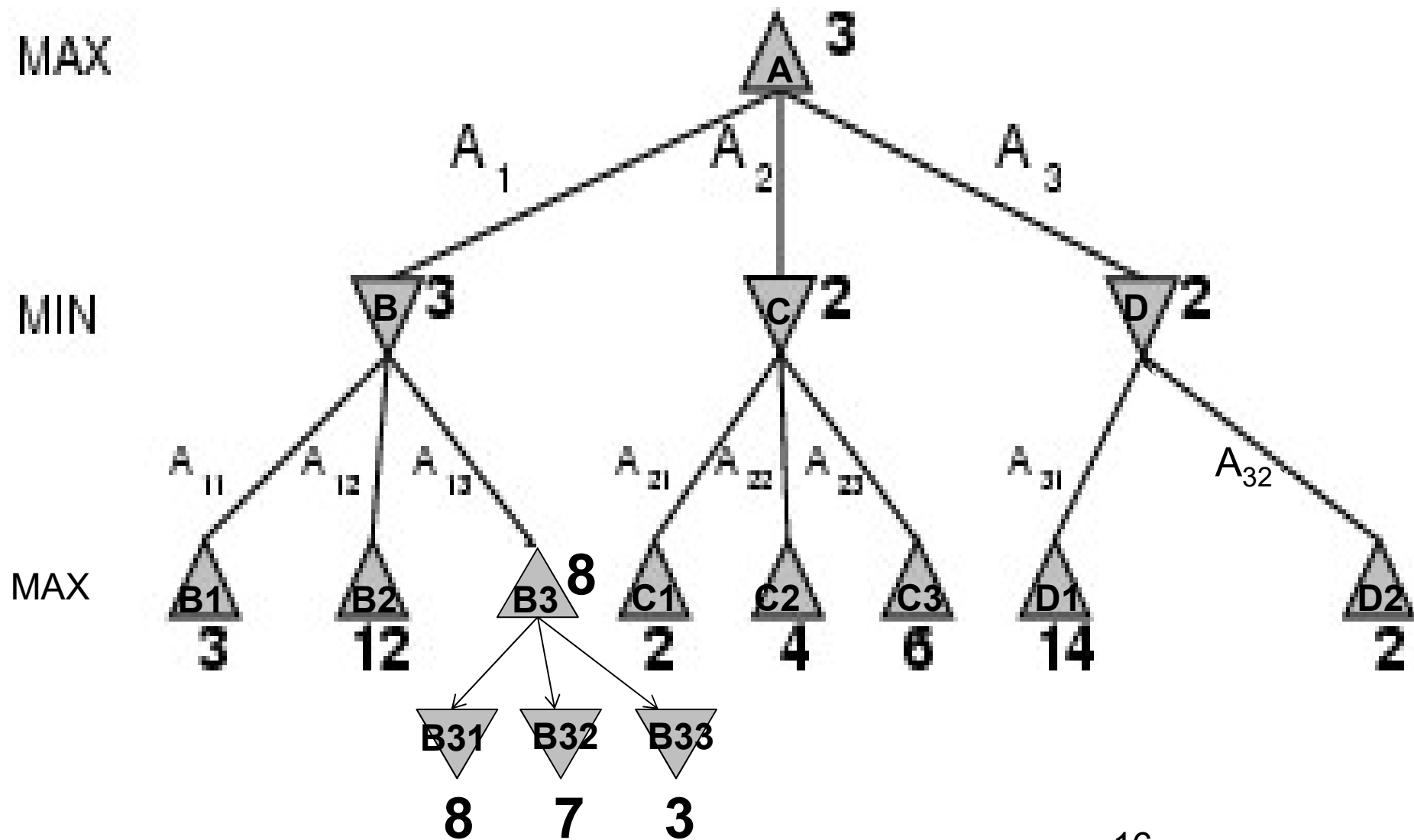
# Ejecución algoritmo: ejemplo

Invocacion maxValue( A ) :  
  . Invocacion minValue( B ) :  
    . . Invocacion maxValue( B1 ) -> que retorna valor terminal 3  
    . . Invocacion maxValue( B2 ) -> que retorna valor terminal 12  
    . . Invocacion maxValue( B3 ) :  
      . . . Invocacion minValue( B31 ) -> que retorna valor terminal 8  
      . . . Invocacion minValue( B32 ) -> que retorna valor terminal 7  
      . . . Invocacion minValue( B33 ) -> que retorna valor terminal 3  
    . . maxValue de B3 asigna valor 8  
  . minValue de B asigna valor 3  
  . Invocacion minValue( C ) :  
    . . Invocacion maxValue( C1 ) -> que retorna valor terminal 2  
    . . Invocacion maxValue( C2 ) -> que retorna valor terminal 4  
    . . Invocacion maxValue( C3 ) -> que retorna valor terminal 6  
  . minValue de C asigna valor 2  
  . Invocacion minValue( D ) :  
    . . Invocacion maxValue( D1 ) -> que retorna valor terminal 14  
    . . Invocacion maxValue( D2 ) -> que retorna valor terminal 2  
  . minValue de D asigna valor 1  
maxValue de A asigna valor 3

valor encontrado: 3

tras el repaso de los sucesores de A la acción a hacer es  $A_1$

# Ejemplo 2 minimax



# Propiedades minimax

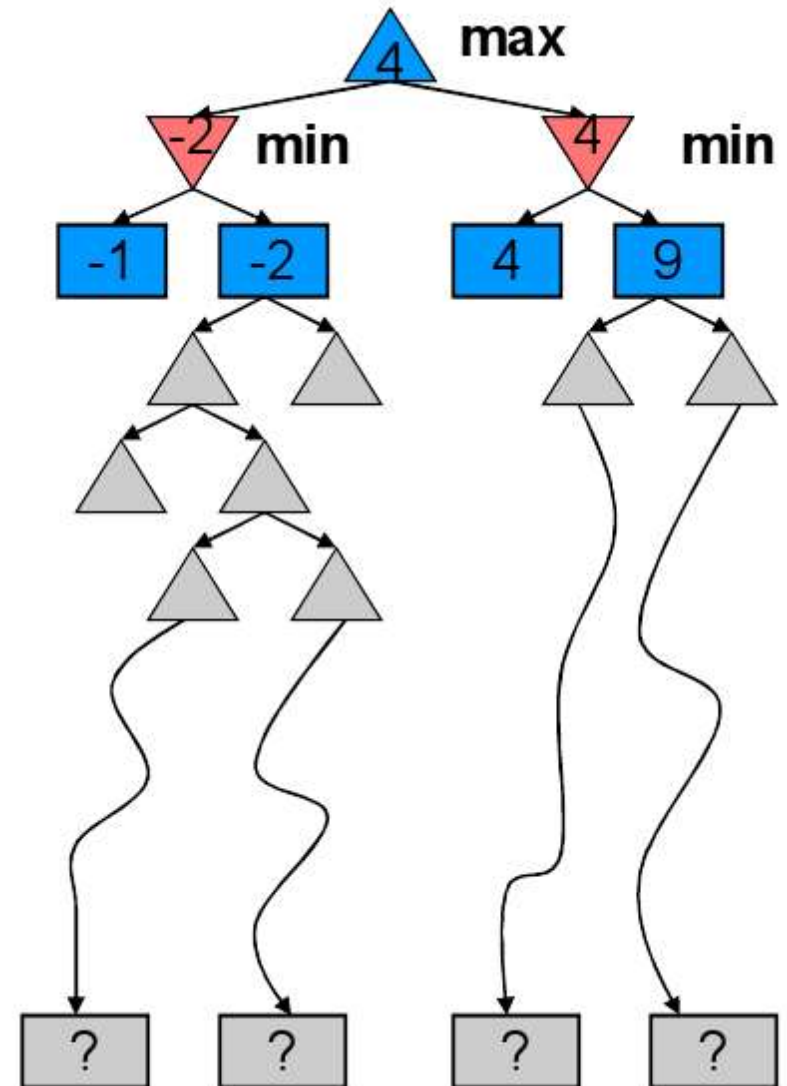
---

- Óptimo contra un enemigo perfecto. ¿Y si no lo es?
  - Complejidad temporal:  $O(b^m)$
  - Complejidad espacial:  $O(b \cdot m)$
  - Completo si el árbol es finito.
- 
- En ajedrez  $b \sim 35$ ,  $m \sim 100 \rightarrow$  La solución exacta es totalmente intratable.

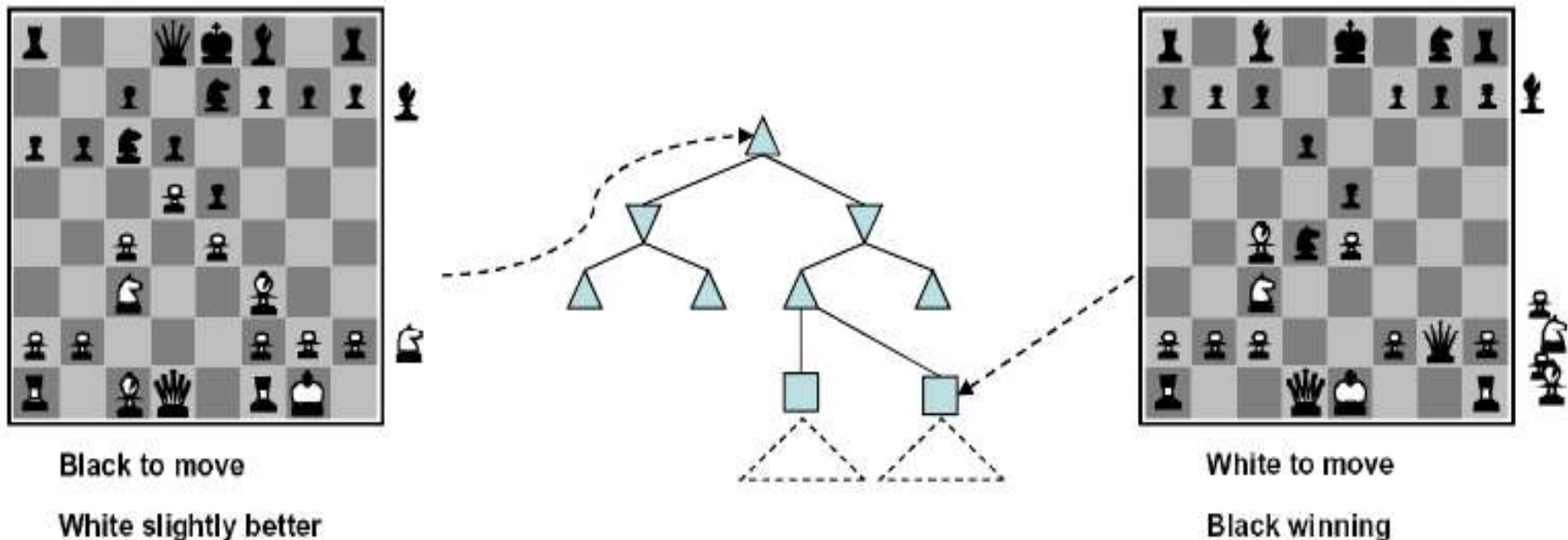


# Reduciendo el gasto computacional

- Buscar hasta una profundidad determinada.
- Reemplazar las utilidades de los nodos terminales por una evaluación heurística.
- La garantía de optimalidad desaparece.
- La profundidad es clave.



# Funciones de evaluación

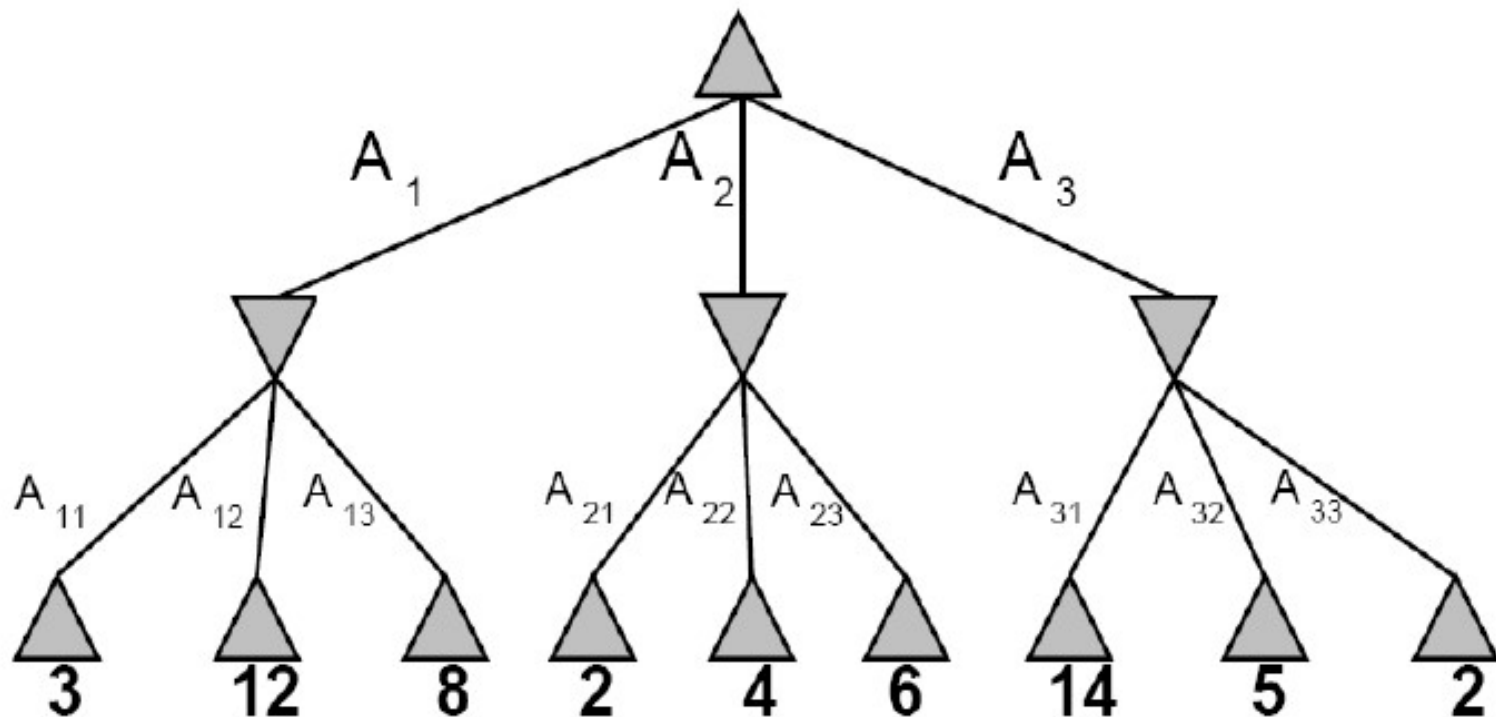


- Debe parecerse lo más posible a la utilidad de esa posición.
- Habitualmente:

$$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \dots + w_n f_n(s)$$

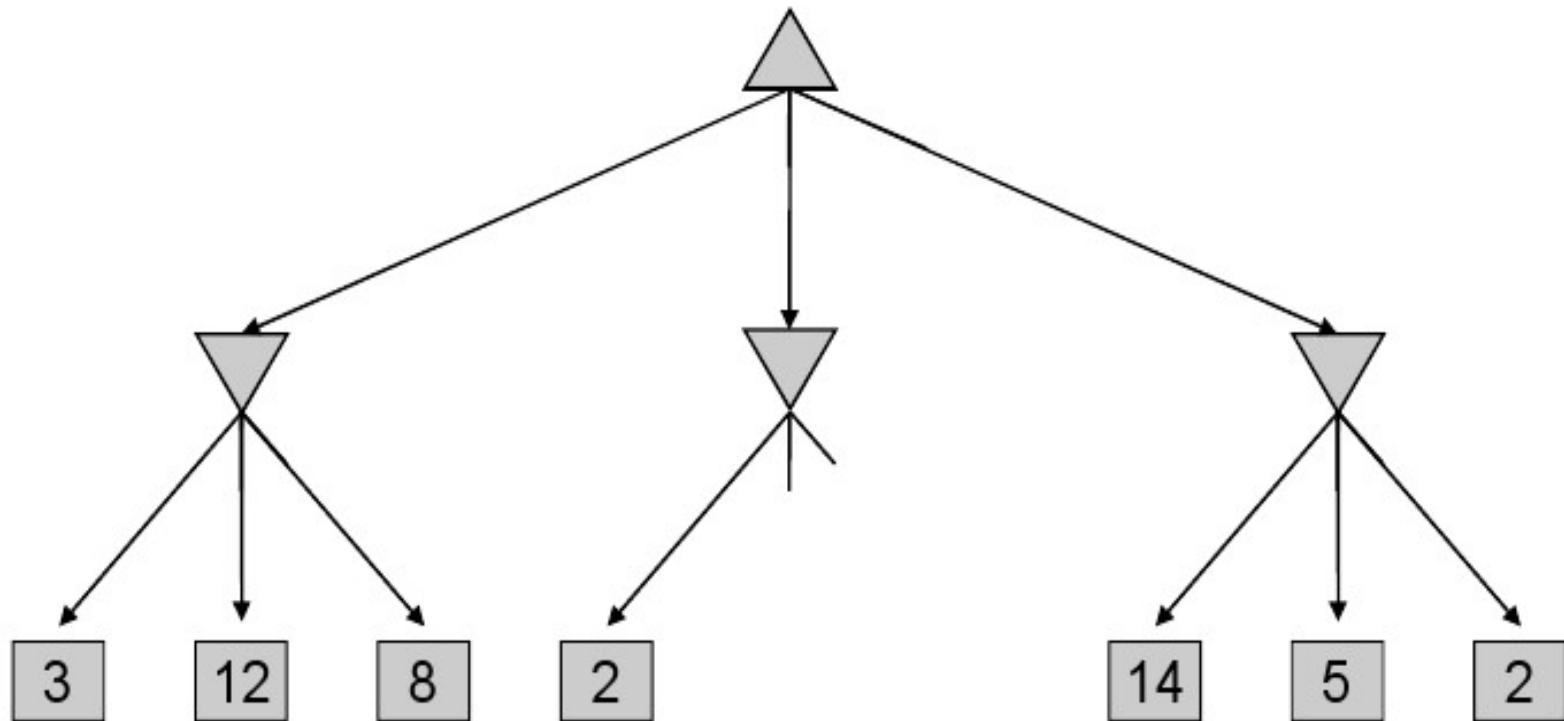
# Podando minimax (ej 1)

- ¿Podemos ahorrarnos expandir o evaluar algún nodo?



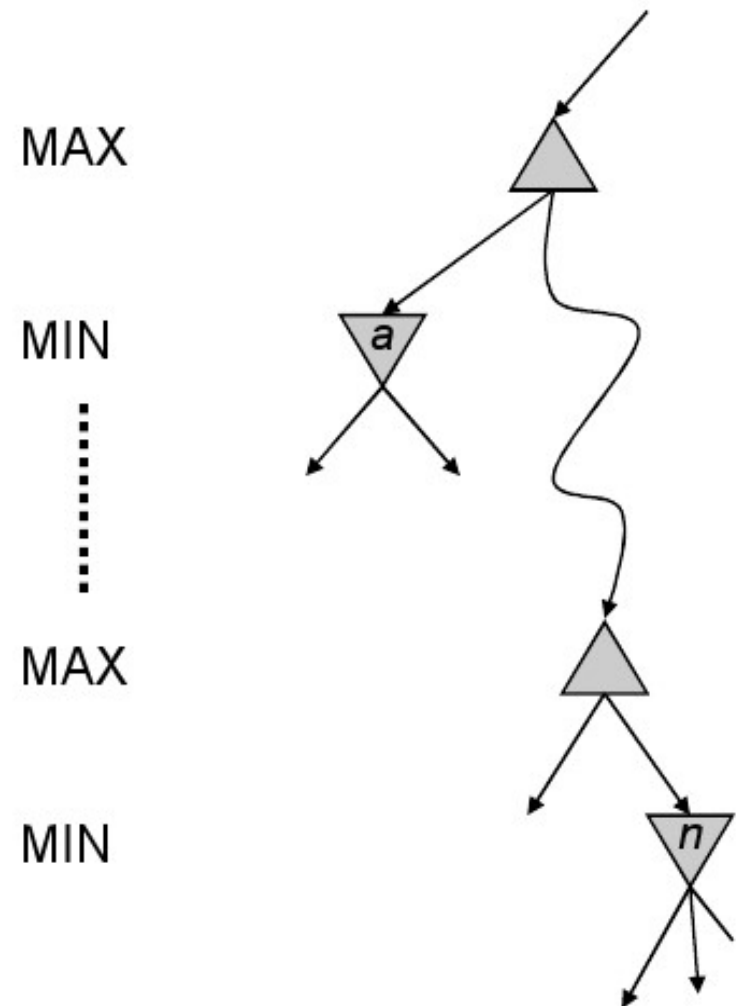
# Podando minimax (ej 1)

- ¿Podemos ahorrarnos expandir o evaluar algún nodo?



# Poda alfa-beta

- Calculando el mínimo en  $n$  (recorriendo los hijos de  $n$ ).
- La estimación del valor de  $n$  va decrementando.
- Sea  $a$  el valor máximo que MAX puede obtener en cualquier elección a lo largo del camino actual.
- Si estimación de  $n < a$  podemos dejar de expandir los hijos de  $n$ , pues tenemos una alternativa mejor.
- Podemos definir de la misma forma  $b$  para MIN.



# Poda alfa-beta

**function** ALPHA-BETA-SEARCH(*state*) *returns an action*

**inputs:** *state*, current state in game

$v \leftarrow \text{MAX-VALUE}(\text{state}, -\infty, +\infty)$

**return** the *action* in SUCCESSORS(*state*) with value *v*

**function** MAX-VALUE(*state*,  $\alpha$ ,  $\beta$ ) *returns a utility value*

**inputs:** *state*, current state in game

$\alpha$ , the value of the best alternative for MAX along the path to *state*

$\beta$ , the value of the best alternative for MIN along the path to *state*

**if** TERMINAL-TEST(*state*) **then return** UTILITY(*state*)

$v \leftarrow -\infty$

**for** *a, s* in SUCCESSORS(*state*) **do**

$v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(s, \alpha, \beta))$

**if**  $v \geq \beta$  **then return** *v*

$\alpha \leftarrow \text{MAX}(\alpha, v)$

**return** *v*

MIN-VALUE (*state*,  $\alpha$ ,  $\beta$ ):

**if** TERMINAL-TEST(*state*) **then return** UTILITY(*state*)

$v \leftarrow +\infty$

**for** *a, s* in SUSCESSORS(*state*) **do**

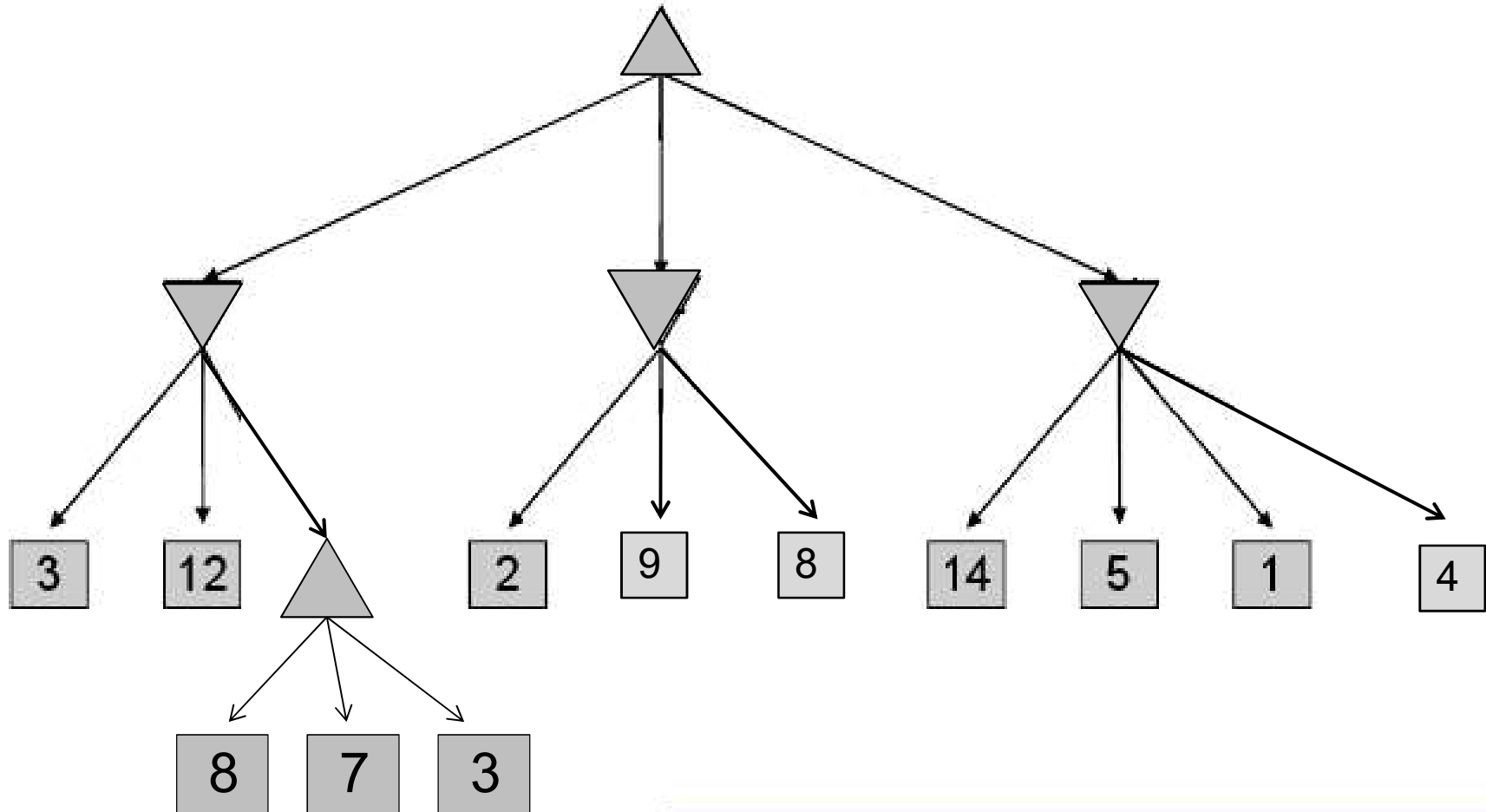
$v \leftarrow \text{Min}(v, \text{MAX-VALUE}(s, \alpha, \beta))$

**if**  $v \leq \alpha$  **then return** *v*

$\beta \leftarrow \text{MIN}(\beta, v)$

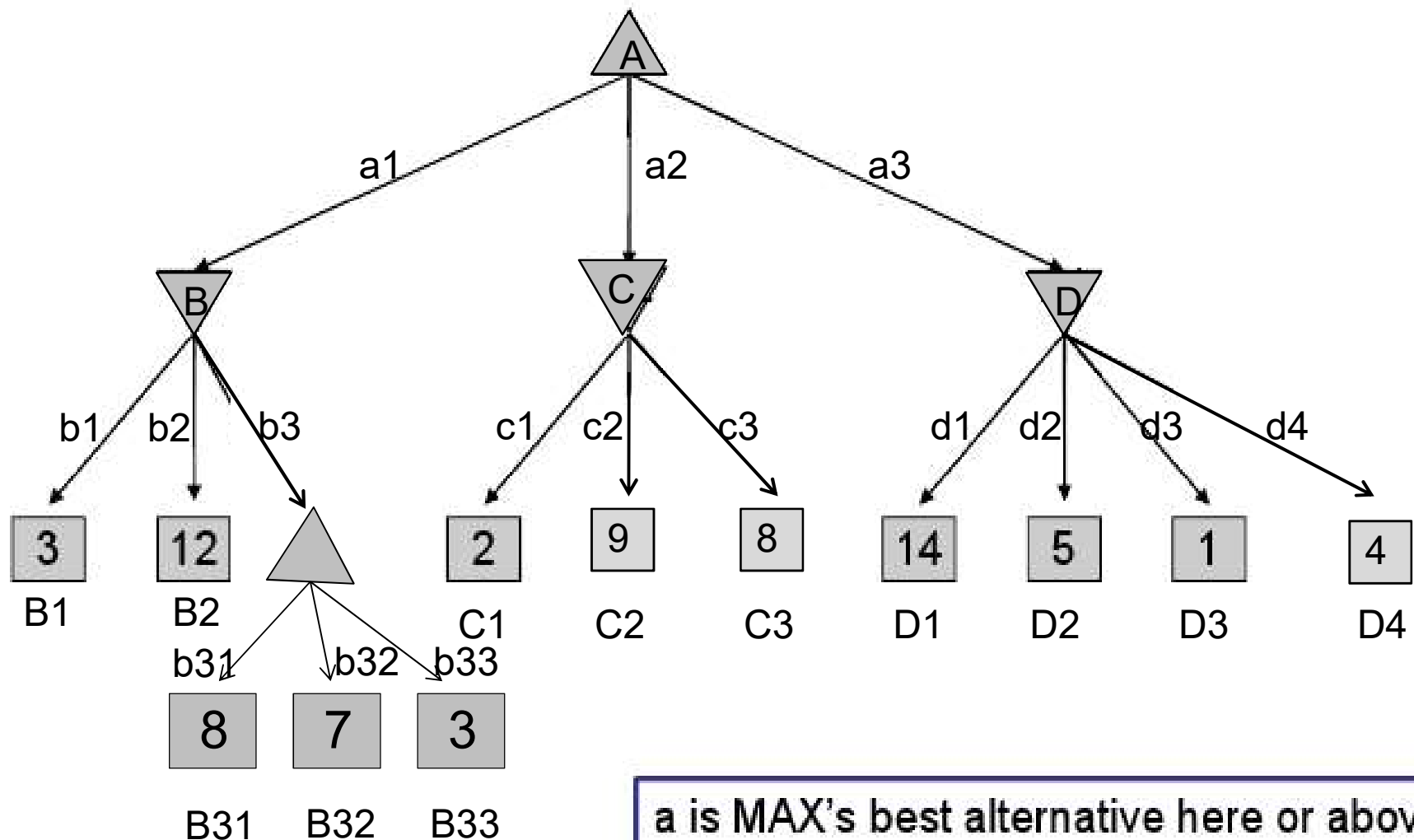
**return** *v*

# Ejemplo poda alfa-beta (ej 3)



a is MAX's best alternative here or above  
b is MIN's best alternative here or above

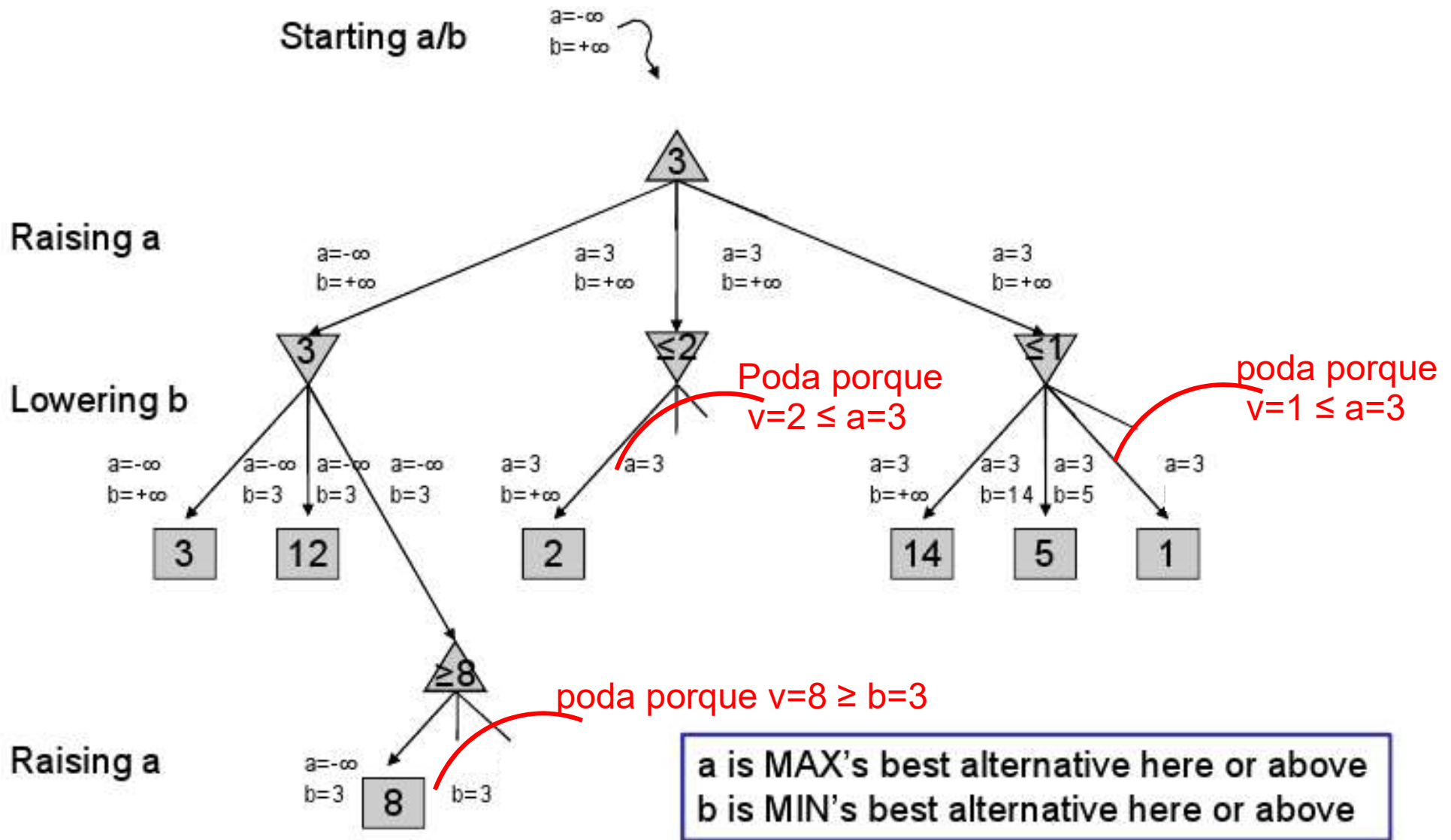
# Ejemplo poda alfa-beta (ej 3)



a is MAX's best alternative here or above  
b is MIN's best alternative here or above



# Ejemplo poda alfa-beta (ej 3)



# Ejecución algoritmo: ejemplo

(denotamos  $\infty$  como inf)

```
Invocacion maxValue( A , alfa= -inf , beta= inf ) :  
  . Invocacion minValue( B , alfa= -inf , beta= inf ) :  
    . . Invocacion maxValue( B1 , alfa= -inf , beta= inf ) -> que retorna valor terminal 3  
    . . Invocacion maxValue( B2 , alfa= -inf , beta= 3 ) -> que retorna valor terminal 12  
    . . Invocacion maxValue( B3 , alfa= -inf , beta= 3 ) :  
      . . . Invocacion minValue( B31 , alfa= -inf , beta= 3 ) -> que retorna valor terminal 8  
      . . PODA en maxValue de B3 porque  $v = 8 \geq \beta = 3$  -> le asigna valor 8  
      . minValue de B asigna valor 3  
    . Invocacion minValue( C , alfa= 3 , beta= inf ) :  
      . . Invocacion maxValue( C1 , alfa= 3 , beta= inf ) -> que retorna valor terminal 2  
      . PODA en minValue de C porque  $v = 2 \leq \alpha = 3$  -> le asigna valor 2  
      . Invocacion minValue( D , alfa= 3 , beta= inf ) :  
        . . Invocacion maxValue( D1 , alfa= 3 , beta= inf ) -> que retorna valor terminal 14  
        . . Invocacion maxValue( D2 , alfa= 3 , beta= 14 ) -> que retorna valor terminal 5  
        . . Invocacion maxValue( D3 , alfa= 3 , beta= 5 ) -> que retorna valor terminal 1  
        . PODA en minValue de D porque  $v = 1 \leq \alpha = 3$  -> le asigna valor 1  
      . maxValue de A asigna valor 3
```

valor encontrado: 3, tras el repaso de los sucesores de A, la acción a hacer es a1

# Propiedades de la poda alfa-beta

- No se sacrifica la optimalidad.
- Una correcta ordenación de los hijos puede mejorar la cantidad de ramas podadas.
- Con una ordenación perfecta se explora el doble de profundidad que sin poda.
- A pesar de todo... el ajedrez se resiste.

# Búsqueda en juegos

---

- Tipos de juegos
- Juegos de suma cero de dos jugadores.
  - Minimax
  - Poda alfa-beta
- Juegos contra la naturaleza.
  - Repaso de probabilidades
  - Expectimax
- Juegos de suma no cero.



# Pregunta

---

- Supongamos que habéis entregado un ejercicio.
- Si a cada uno de vosotros os pregunto “qué forma de evaluar el ejercicio prefieres?”:
- a) Elegir por sorteo 12 ejercicios y a esos les pongo un 10 de nota y al resto 5.
  - b) Poner a esos 12 un 8 y al resto un 7.
  - c) Elegir los 12 mejores ejercicios, ponerles un 10 y al resto un 5
  - d) Poner a esos 12 un 8 y al resto un 7.

# Pregunta

Supongamos que habéis entregado un ejercicio  
Si a cada uno de vosotros os pregunto “qué forma de evaluar el ejercicio prefieres?”:

- a) Elegir por sorteo 12 ejercicios y a esos les pongo un 10 de nota y al resto 5
- b) Poner a esos 12 un 8 y al resto un 7.  SI ej. malo
- c) Elegir los 12 mejores ejercicios, ponerles un 10 y al resto un 5  SI has hecho un buen ejercicio
- d) Poner a esos 12 un 8 y al resto un 7.

Cálculo?

# Pregunta: Cálculo

---

Si a cada uno de vosotros os pregunto “qué forma de evaluar el ejercicio prefieres?”:

- a) Elegir por sorteo 12 ejercicios y a esos les pongo un 10 de nota y al resto 5.
- b) Poner a esos 12 un 8 y al resto un 7.
- c) Elegir los 12 mejores ejercicios, ponerles un 10 y al resto un 5 (has hecho un buen trabajo)
- d) Poner a esos 12 un 8 y al resto un 7.

Cálculo:

$$E(\text{nota}) = P(\text{eleg}) * \text{nota\_eleg} + (1 - P(\text{eleg})) * \text{nota\_no\_e}$$

# Pregunta: Valores Respuesta

“Qué forma de evaluar los ejercicios prefieres?”:

- a) Elegir por sorteo 12 ejercicios y a esos les pongo un 10 de nota y al resto 5
- b) Poner a esos 12 un 8 y al resto un 7.
- c) Elegir los 12 mejores ejercicios, ponerles un 10 y al resto un 5 (has hecho un buen trabajo)
- d) Poner a esos 12 un 8 y al resto un 7.

Asumimos que lo han entregado 50:

$$E(\text{nota}) = P(\text{eleg}) * \text{nota\_eleg} + (1 - P(\text{eleg})) * \text{nota\_no\_e}$$

a)  $P(\text{elegido}) = 12/50 \approx 0.25$ ;  $0.25 * 10 + 0.75 * 5 = 6.25$

b)  $0.25 * 8 + 0.75 * 7 = 7.25$

c)  $P(\text{elegido}) = 0.75$  (buen ej);  $0.75 * 10 + 0.25 * 5 = 8.75$

d)  $0.75 * 8 + 0.25 * 7 = 7.75$



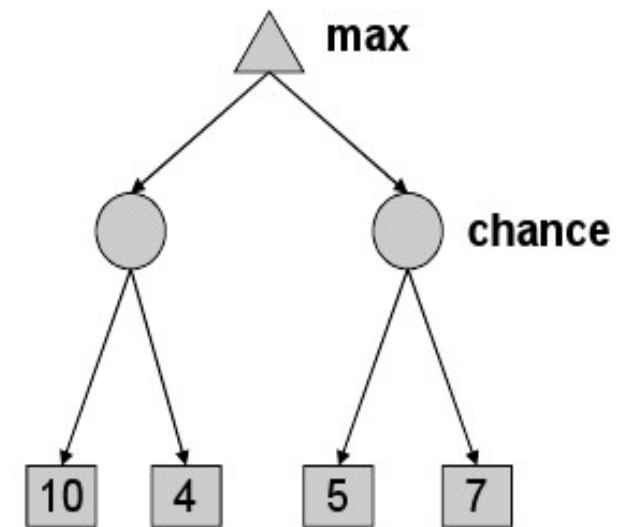
# Búsqueda expectimax

---

- ¿Qué pasa si no conocemos cuál será el resultado de una acción? (Problema no determinista)
  - Pedir una carta en el solitario.
  - Probar una mina en el buscaminas.
  - Pacman contra fantasmas aleatorios.
- Podemos usar búsqueda expectimax:
  - Los nodos del "enemigo" en lugar de calcular el mínimo calcula la esperanza (el valor medio).
- Más adelante → Markov Decision Processes.

# Utilidad máxima esperada

- ¿Por qué no usamos minimax?
- Principio de máxima utilidad esperada: Un agente debe seleccionar la acción que **maximice su utilidad esperada dado su conocimiento.**
- Principio general para la toma de decisiones.
- Definición de racionalidad.
- Aparecerá más a lo largo del curso.



# Probabilidades

---

- Una **variable aleatoria** representa un hecho cuyo resultado desconocemos.
- Una **distribución de probabilidad** es una asignación de pesos a los diferentes eventos del hecho.
- Ejemplo: Tráfico en la autopista
  - Variable aleatoria:  $T$  = cuánto tráfico hay.
  - Eventos: ligero, normal, denso.
  - Distribución:  $P(\{T=\text{ligero}\}) = 0.4$   $P(\{T=\text{normal}\})=0.5$   
 $P(\{T=\text{denso}\})=0.1$

# Probabilidades

---

- Las probabilidades son siempre no negativas.
- Las probabilidades sobre todos los eventos posibles suman 1.
- A medida que tenemos más información, las probabilidades cambian.
  - $P(\{T=\text{denso}\})=0.1$
  - $P(\{T=\text{denso}\}|\{hora=8am\})=0.4$
- Más adelante veremos métodos para razonar y actualizar las probabilidades.

# ¿Qué son las probabilidades?

---

- Visión frecuentista:
  - Medias sobre experimentos repetidos.
  - Se estiman a partir de observaciones históricas.
  - Nos permiten saber cómo funcionarán los experimentos futuros (a largo plazo).
  - Nos hace pensar en hechos *inherentemente aleatorios* como lanzar dados.
- Visión Bayesiana:
  - Grados de creencia sobre variables no observadas (en base al conocimiento).
    - La creencia de un agente en que está lloviendo, dada la temperatura.
    - La creencia de pacman de que un fantasma va a girar, dado el estado.
  - Se pueden *aprender* a partir de la experiencia, ya que las nuevas experiencias modifican nuestras creencias.

# Incertidumbre por todos lados

---

- No sólo en juegos de azar:
  - Estoy resfriado. ¿Estornudaré en el próximo minuto?
  - Este correo contiene "Viagra". ¿Es spam?
  - Me duele un diente. ¿Tiene caries?
  - ¿Llegaré en 30 min al aeropuerto?
  - El robot ha hecho girar la rueda tres vueltas. ¿Cuánto se ha desplazado?
  - ¿Es seguro cruzar la calle ahora?
- Fuentes de incertidumbre:
  - Procesos aleatorios: dados...
  - Información insuficiente.
  - Ignorancia de los procesos subyacentes.
  - Variables que no se incluyen en el modelo.
  - El mundo es ruidoso.

# Recordatorio: Esperanzas

---

- Podemos definir una función  $f(X)$  de una variable aleatoria  $X$ .
- El valor esperado de una función es su valor medio, ponderando cada valor de su entrada por la distribución de probabilidad.
- Ejemplo: ¿Cuánto tarda en llegar al aeropuerto?
  - Es función del tráfico:
    - $f(\text{ligero})=10$ ,  $f(\text{normal})=15$ ,  $f(\text{denso})=40$
    - $P(\{T=\text{ligero}\}) = 0.4$   $P(\{T=\text{normal}\})=0.5$   $P(\{T=\text{denso}\})=0.1$
  - ¿Cuál es el tiempo esperado?  $E[f(T)]$

# Recordatorio: Esperanzas

- Podemos definir una función  $f(X)$  de una variable aleatoria  $X$ .
- El valor esperado de una función es su valor medio, ponderando cada valor de su entrada por la distribución de probabilidad.
- Ejemplo: ¿Cuánto tarda en llegar al aeropuerto?
  - Es función del tráfico:
    - $f(\text{ligero})=10$ ,  $f(\text{normal})=15$ ,  $f(\text{denso})=40$
    - $P(\{T=\text{ligero}\}) = 0.4$   $P(\{T=\text{normal}\})=0.5$   $P(\{T=\text{denso}\})=0.1$
  - ¿Cuál es el tiempo esperado?  $E[f(T)]$   
 $E[f(T)] = P(\text{ligero}) \cdot f(\text{ligero}) + P(\text{normal}) \cdot f(\text{normal}) + P(\text{denso}) \cdot f(\text{denso})$   
 $E[f(T)] = 0.4 \cdot 10 + 0.5 \cdot 15 + 0.1 \cdot 40 = 4 + 7.5 + 4 = 15.5$



# Esperanzas

- Esperanza de una función de una variable aleatoria:

$$E_{P(X)}[f(X)] = \sum_x f(x)P(x)$$

- Ejemplo: ¿Cuál es el valor medio de un dado?

$X$	$P$	$f$
1		1
2		2
3		3
4		4
5		5
6		6

# Esperanzas

- Esperanza de una función de una variable aleatoria:

$$E_{P(X)}[f(X)] = \sum_x f(x)P(x)$$

- Ejemplo: ¿Cuál es el valor medio de un dado?

$X$	$P$	$f$
1	1/6	1
2	1/6	2
3	1/6	3
4	1/6	4
5	1/6	5
6	1/6	6

# Esperanzas

- Esperanza de una función de una variable aleatoria:

$$E_{P(X)}[f(X)] = \sum_x f(x)P(x)$$

- Ejemplo: valor medio de un dado.

$X$	$P$	$f$
1	1/6	1
2	1/6	2
3	1/6	3
4	1/6	4
5	1/6	5
6	1/6	6

$$1 \times \frac{1}{6} + 2 \times \frac{1}{6} + 3 \times \frac{1}{6} + 4 \times \frac{1}{6} + 5 \times \frac{1}{6} + 6 \times \frac{1}{6} \\ = 3.5$$

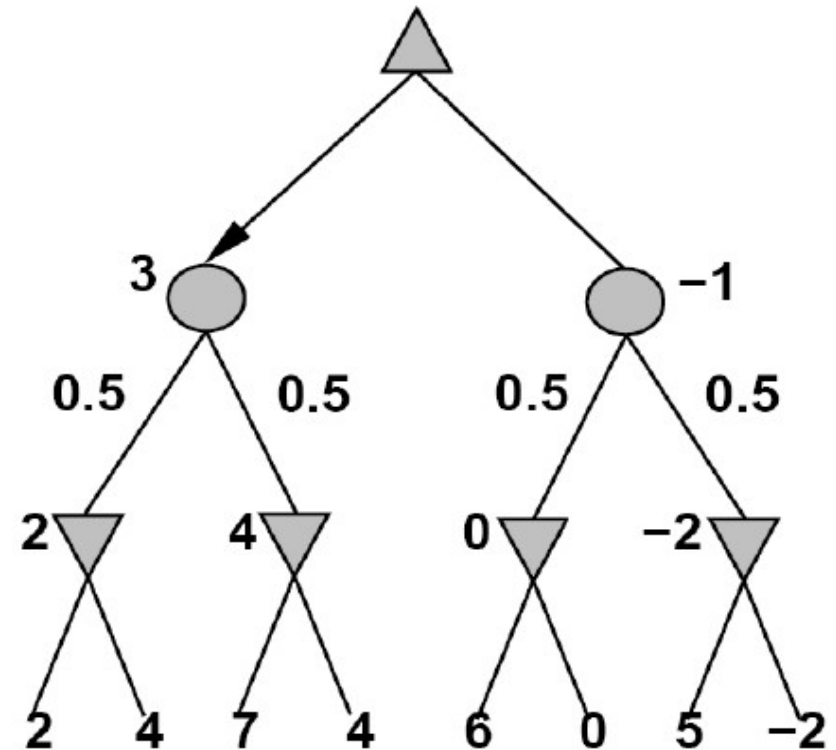
# Utilidades

---

- Las utilidades son funciones que van de los resultados (estados del mundo) a números reales que representan las preferencias de un agente.
- ¿De dónde salen las utilidades?
  - En un juego, sencillo: ganar= +1, perder =-1
  - Las utilidades resumen los objetivos de un agente.
  - Teorema: Cualquier conjunto de preferencias coherente se puede representar mediante una función de utilidad.

# Expectiminimax

- Expectiminimax:
  - El entorno es un jugador especial que juega después de cada jugador.
  - Los nodos aleatorios usan la esperanza y el resto como minimax.



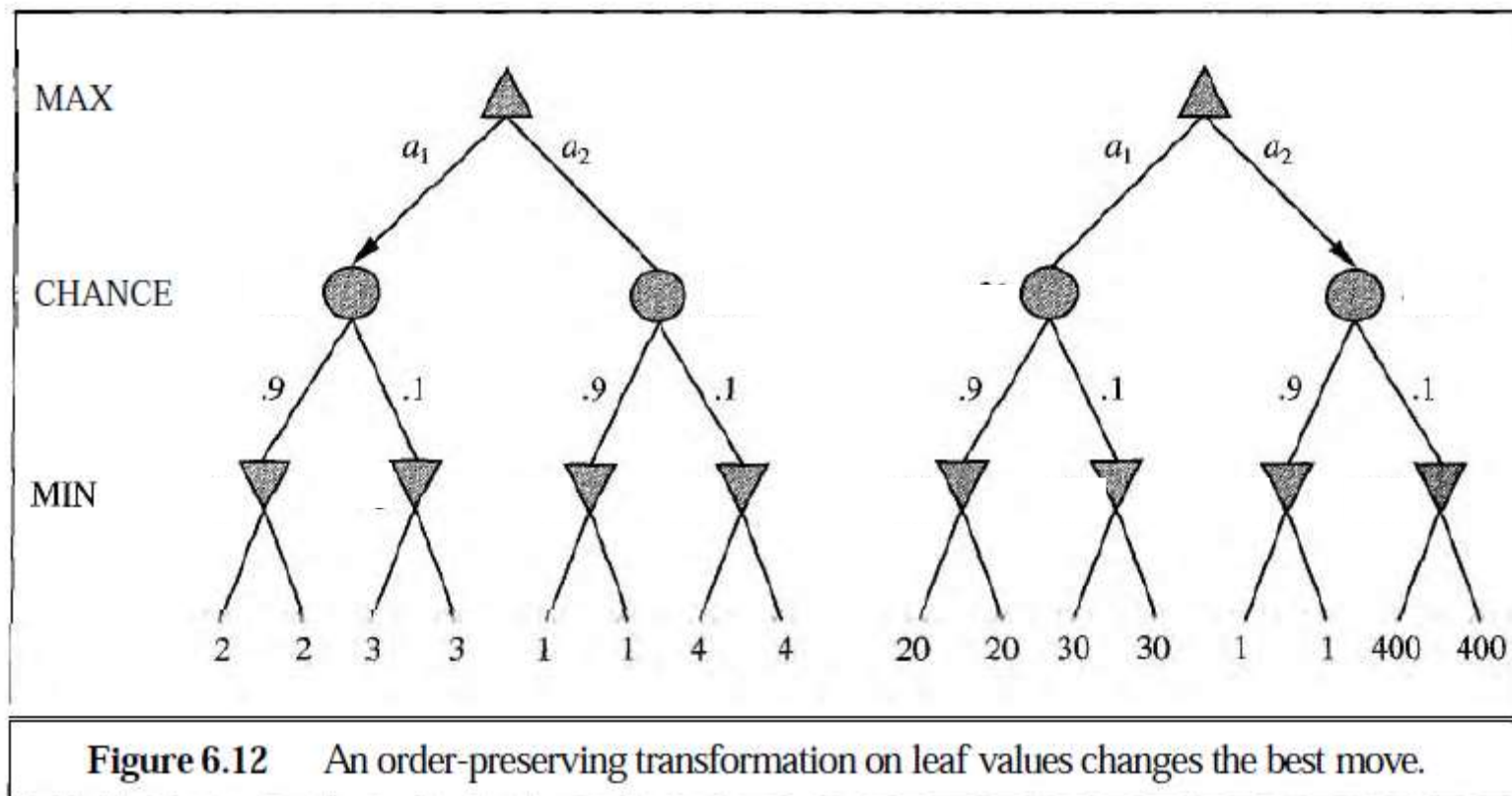
EXPECTIMINIMAX( $n$ ) =

$$\begin{cases} \text{UTILITY}(n) & \text{if } n \text{ is a terminal state} \\ \max_{s \in \text{Successors}(n)} \text{EXPECTIMINIMAX}(s) & \text{if } n \text{ is a MAX node} \\ \min_{s \in \text{Successors}(n)} \text{EXPECTIMINIMAX}(s) & \text{if } n \text{ is a MIN node} \\ \sum_{s \in \text{Successors}(n)} P(s) \cdot \text{EXPECTIMINIMAX}(s) & \text{if } n \text{ is a chance node} \end{cases}$$

- Ej: Backgammon

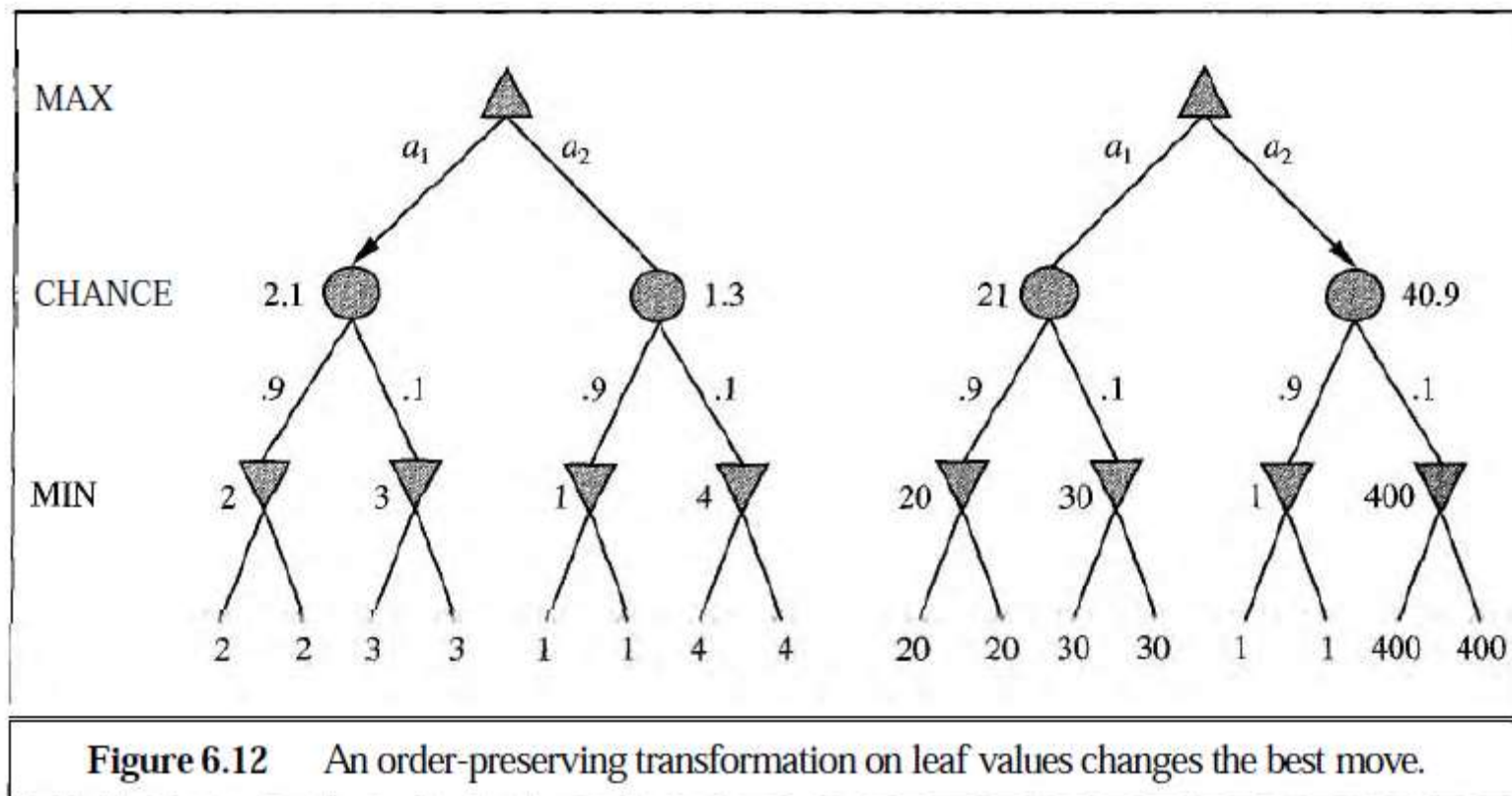
# Funciones de evaluación en expectiminimax

- En minimax el valor de las funciones de evaluación no importa, tan sólo es importante el orden relativo (si un estado es mejor que otro o no)
- Para expectiminimax necesitamos además que las magnitudes de los valores sean correctas.



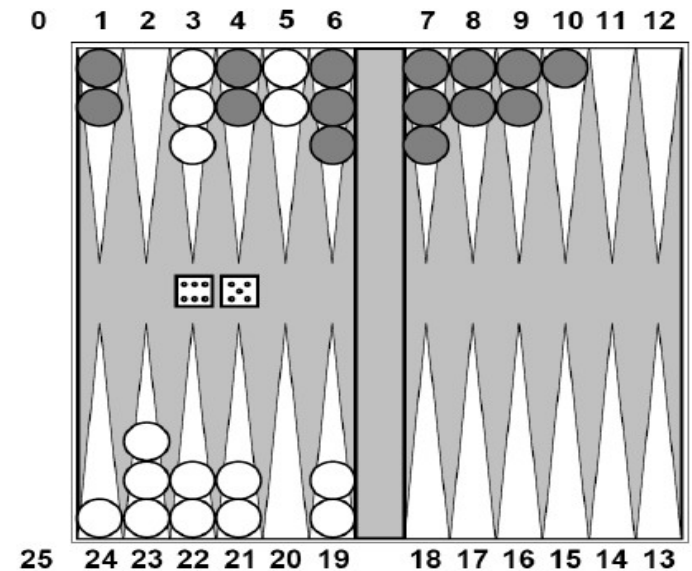
# Funciones de evaluación en expectiminimax

- En minimax el valor de las funciones de evaluación no importa, tan sólo es importante el orden relativo (si un estado es mejor que otro o no)
- Para expectiminimax necesitamos además que las magnitudes de los valores sean correctas.



# Backgammon

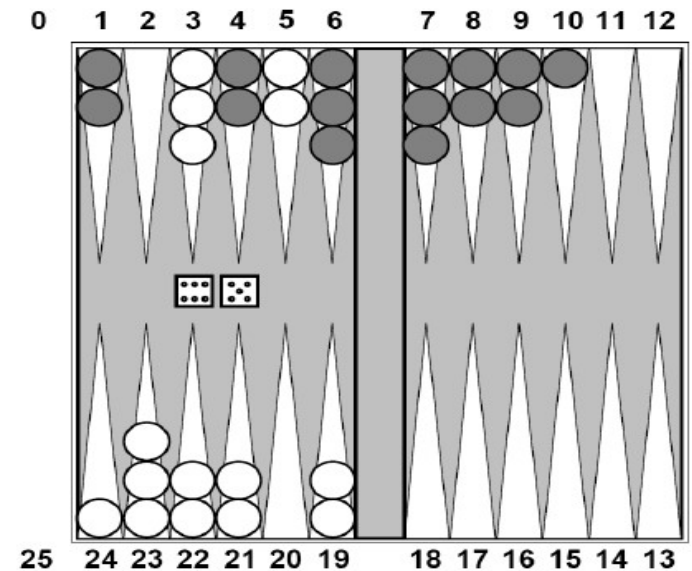
- TD-Gammon:
  - <http://en.wikipedia.org/wiki/TD-Gammon>
  - Búsqueda a 2 niveles + buena función de evaluación: Temporal Difference learning (red neuronal)
  - Al nivel del campeón del mundo.
- El lanzamiento de dados incrementa b:
  - posibilidades diferentes con 2 dados ?





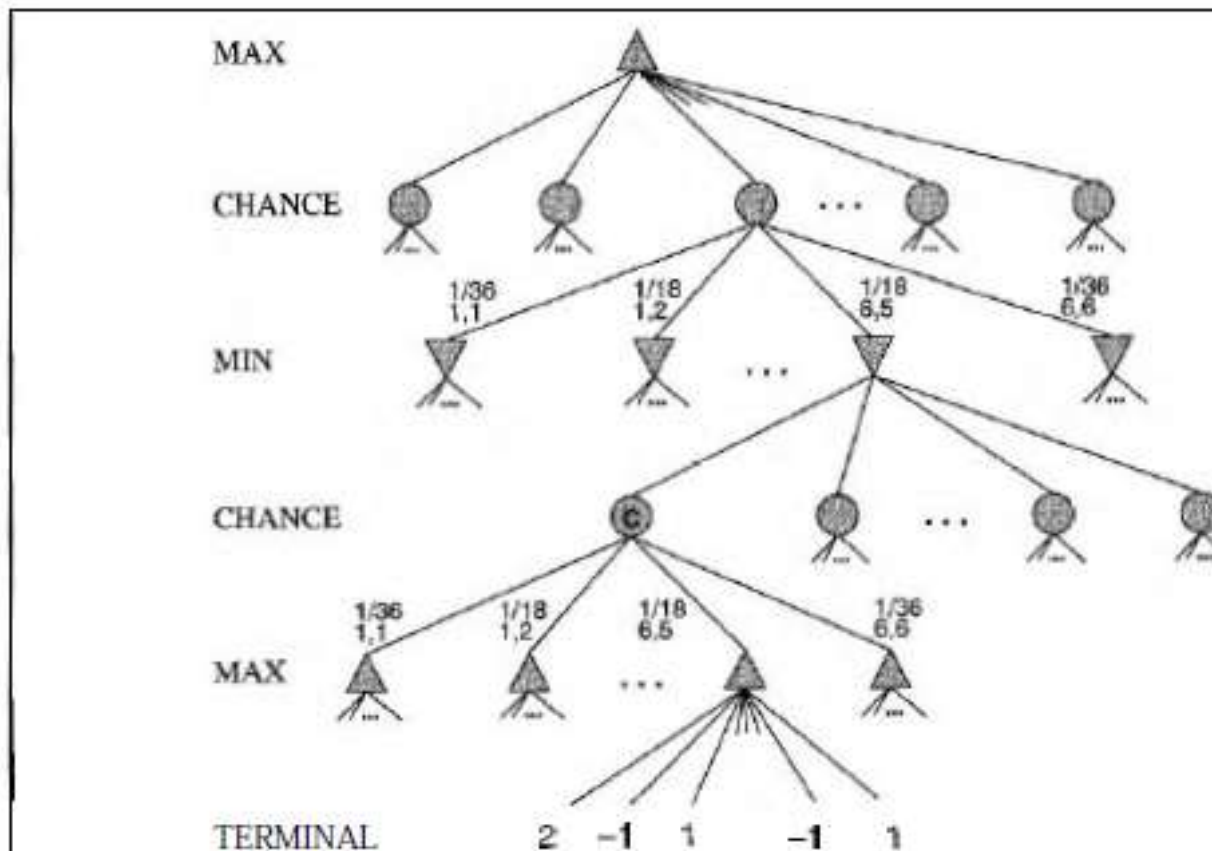
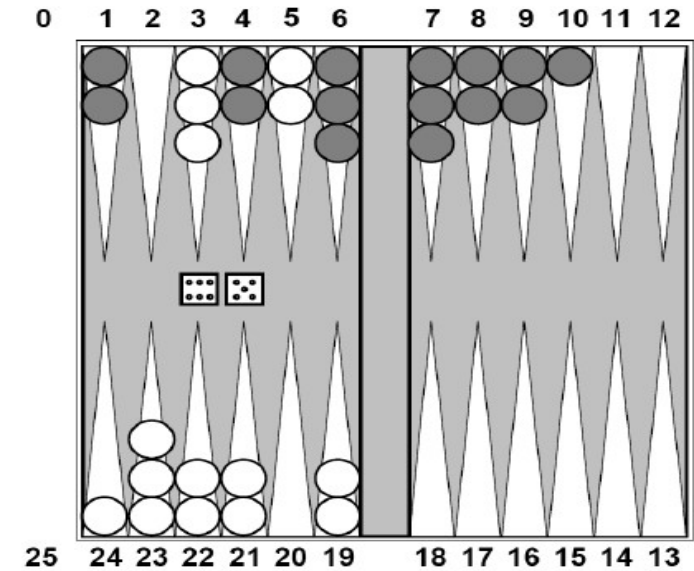
# Backgammon

- TD-Gammon:
  - <http://en.wikipedia.org/wiki/TD-Gammon>
  - Búsqueda a 2 niveles + buena función de evaluación: Temporal Difference learning (red neuronal)
  - Al nivel del campeón del mundo.
- El lanzamiento de dados incrementa b:
  - posibilidades diferentes con 2 dados: **21**



# Backgammon

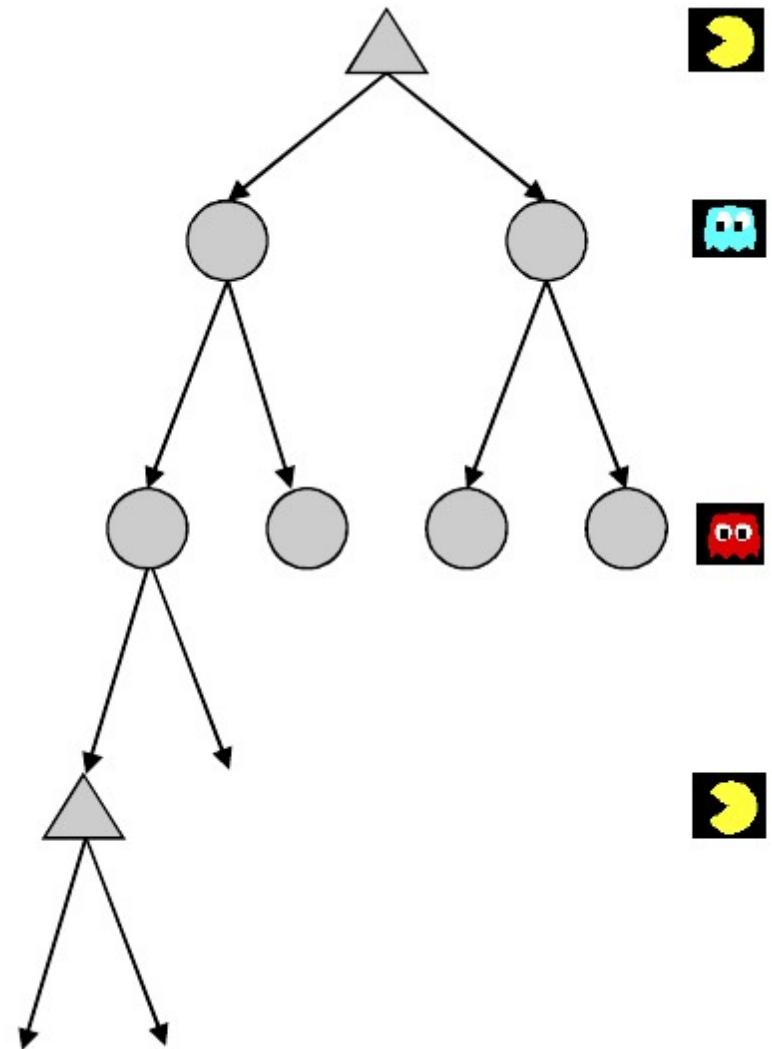
- A medida que bajamos niveles disminuye la probabilidad de que lleguemos a esa configuración.



**Figure 6.11** Schematic game tree for a backgammon position.

# Búsqueda **expectimax**

- Tenemos un modelo probabilístico de cómo se comportan los rivales:
  - Sencillo: Lanzar un dado.
  - Complejo.
  - Un nodo representa cada hecho fuera de nuestro control (enemigo o naturaleza).
  - El modelo podría decir que el agente se comporta como un adversario ideal.

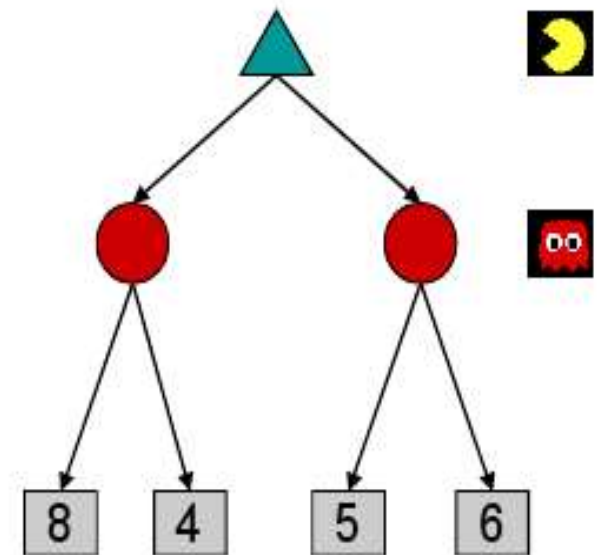


# Pseudocódigo para expectimax

```
def value(s)
  if s is a max node return maxValue(s)
  if s is an exp node return expValue(s)
  if s is a terminal node return evaluation(s)
```

```
def maxValue(s)
  values = [value(s') for s' in successors(s)]
  return max(values)
```

```
def expValue(s)
  values = [value(s') for s' in successors(s)]
  weights = [probability(s, s') for s' in successors(s)]
  return expectation(values, weights)
```

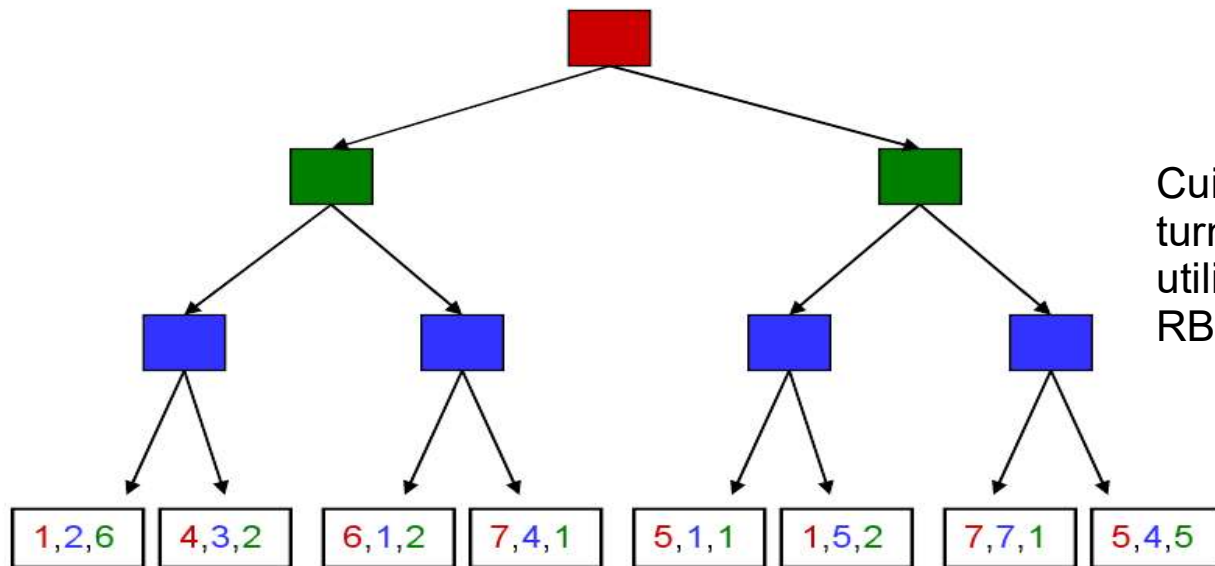


# Expectimax para Pacman

---

- Nos hemos alejado de la idea de que los fantasmas tratan de ganarnos el juego. Ahora los consideramos "parte del entorno".
- Pacman tiene una distribución de creencias sobre como se comportarán.
- ¿Podemos ver minimax como un caso especial de expectimax?
- ¿Cómo se comportaría Pacman si asumiera que los fantasmas realizan minimax a un nivel el 80% del tiempo y juegan aleatoriamente el resto del tiempo?
- Podemos modelar al rival pensando que él tiene un modelo de nosotros...

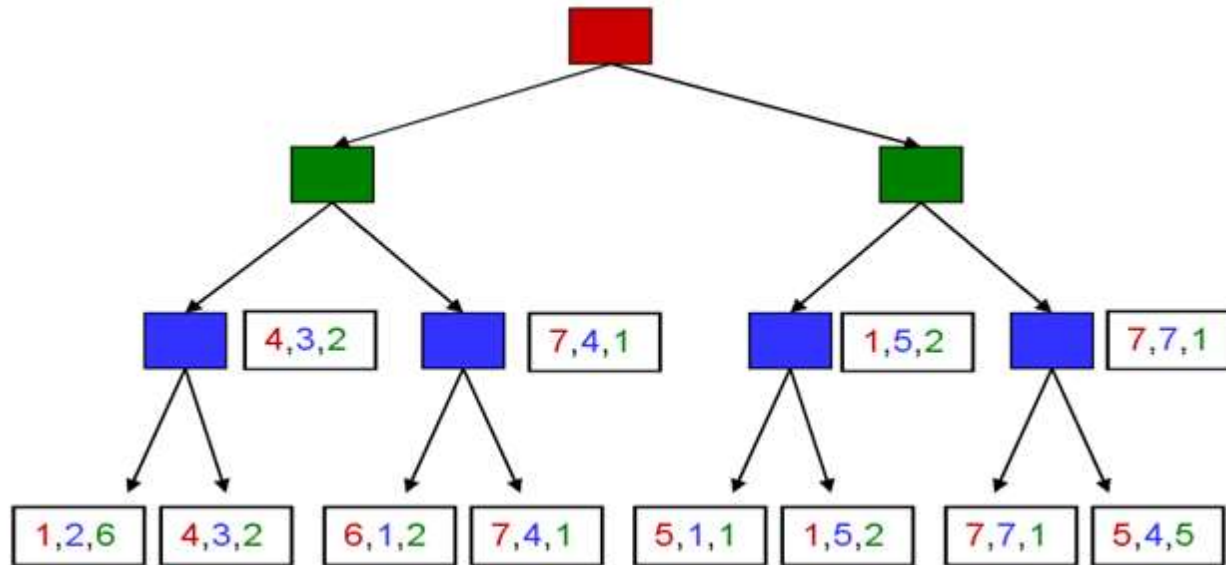
# Juegos de suma no cero o multijugador



Cuidado con el orden:  
turnos R,G,B pero  
utilidades descritas como  
RBG

- Similar al minimax:
  - Las utilidades son ahora tuplas.
  - Cada jugador **maximiza** su propia entrada y propaga el resultado al siguiente nivel.
  - Diplomacy game

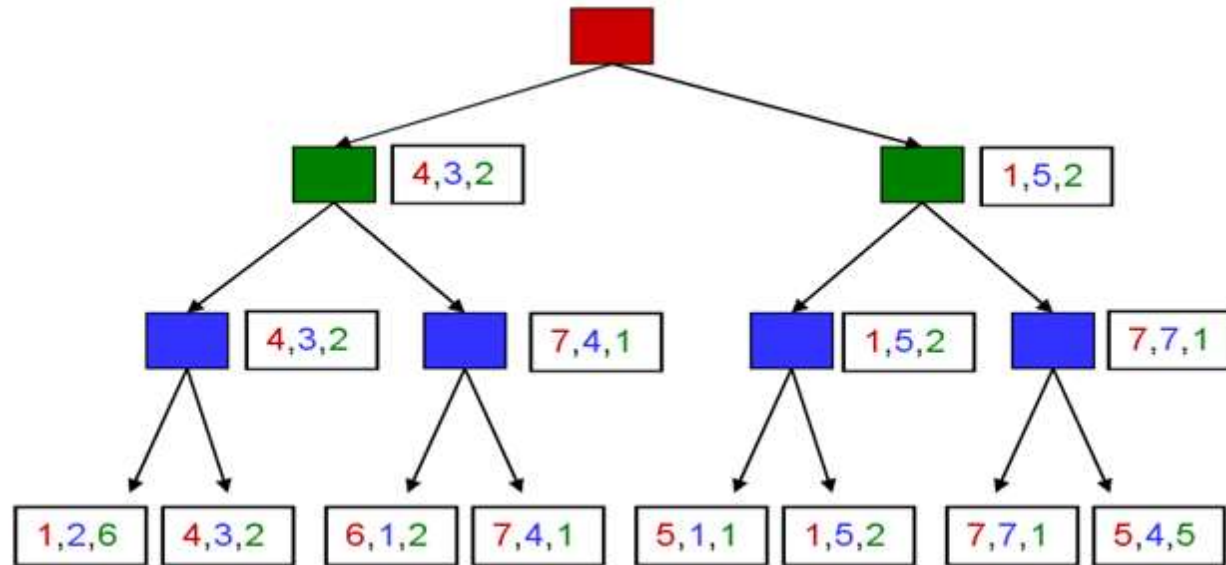
# Juegos de suma no cero o multijugador



- Similar al minimax:
  - Las utilidades son ahora tuplas.
  - Cada jugador **maximiza** su propia entrada y propaga el resultado al siguiente nivel.
  - Diplomacy game



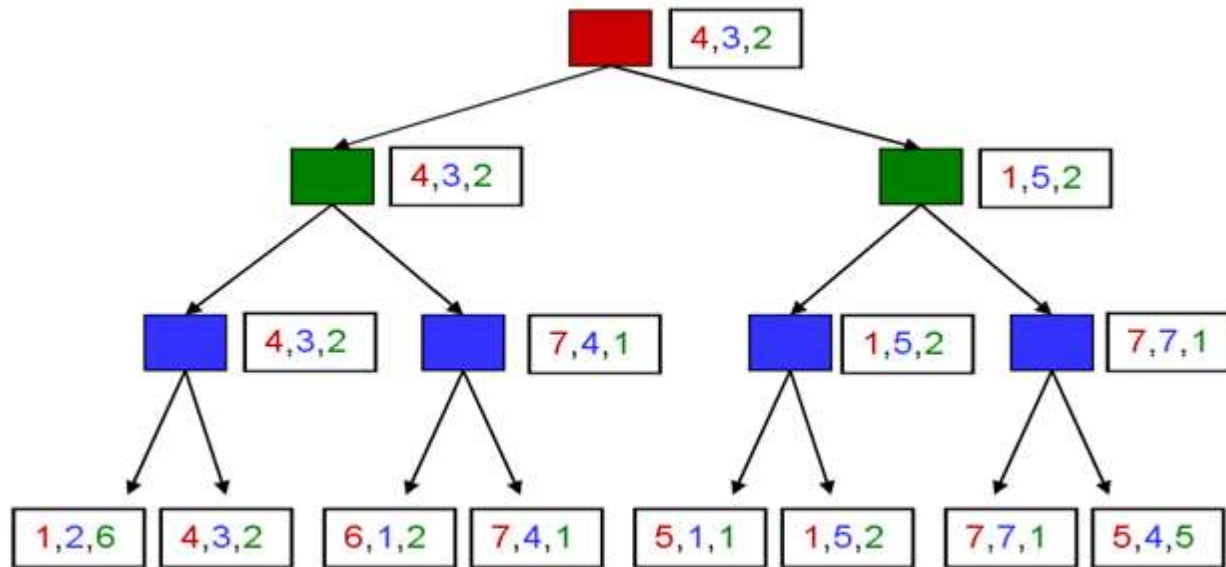
# Juegos de suma no cero o multijugador



- Similar al minimax:
  - Las utilidades son ahora tuplas.
  - Cada jugador **maximiza** su propia entrada y propaga el resultado al siguiente nivel.
  - Diplomacy game



# Juegos de suma no cero o multijugador



- Similar al minimax:
  - Las utilidades son ahora tuplas.
  - Cada jugador **maximiza** su propia entrada y propaga el resultado al siguiente nivel.
  - Diplomacy game

# Repaso algoritmos

---

- Minimax
- Poda alfa-beta
- Expectiminimax

# Algoritmo minimax (T<sup>a</sup>)

**function** MINIMAX-DECISION(*state*) *returns an action*

$v \leftarrow \text{MAX-VALUE}(\textit{state})$

**return** the *action* in SUCCESSORS(*state*) with value *v*

---

**function** MAX-VALUE(*state*) *returns a utility value*

**if** TERMINAL-TEST(*state*) **then return** UTILITY(*state*)

$v \leftarrow -\infty$

**for** *a, s* in SUCCESSORS(*state*) **do**

$v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(s))$

**return** *v*

---

**function** MIN-VALUE(*state*) *returns a utility value*

**if** TERMINAL-TEST(*state*) **then return** UTILITY(*state*)

$v \leftarrow \infty$

**for** *a, s* in SUCCESSORS(*state*) **do**

$v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(s))$

**return** *v*

# Algoritmo minimax (Práct)

```
function minimax(nodo, profundidad, deboMaximizar)
  if profundidad = 0 or nodo es terminal
    return evaluación heurística del nodo
  if deboMaximizar
     $\alpha := -\infty$ 
    for each siguiente acción posible
       $\alpha := \max(\alpha, \text{minimax}(\text{hijo}, \text{profundidad} - 1, \text{False}))$ 
    return  $\alpha$ 
  else
     $\beta := +\infty$ ,
    for each siguiente acción posible
       $\beta := \min(\beta, \text{minimax}(\text{hijo}, \text{profundidad} - 1, \text{True}))$ 
    return  $\beta$ 

(* Llamada inicial *)
minimax(origen, profundidad, True)
```

# Algoritmo Poda $\alpha$ - $\beta$ ( $T^a$ )

**function** ALPHA-BETA-SEARCH(*state*) *returns an action*

**inputs:** *state*, current state in game

$v \leftarrow \text{MAX-VALUE}(\text{state}, -\infty, +\infty)$

**return** the *action* in SUCCESSORS(*state*) with value  $v$

**function** MAX-VALUE(*state*,  $\alpha$ ,  $\beta$ ) *returns a utility value*

**inputs:** *state*, current state in game

$\alpha$ , the value of the best alternative for MAX along the path to *state*

$\beta$ , the value of the best alternative for MIN along the path to *state*

**if** TERMINAL-TEST(*state*) **then return** UTILITY(*state*)

$v \leftarrow -\infty$

**for**  $a, s$  in SUCCESSORS(*state*) **do**

$v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(s, \alpha, \beta))$

**if**  $v \geq \beta$  **then return**  $v$

$\alpha \leftarrow \text{MAX}(\alpha, v)$

**return**  $v$

MIN-VALUE (*state*,  $\alpha$ ,  $\beta$ ):

**if** TERMINAL-TEST(*state*) **then return** UTILITY(*state*)

$v \leftarrow +\infty$

**for**  $a, s$  in SUCCESSORS(*state*) **do**

$v \leftarrow \text{Min}(v, \text{MAX-VALUE}(s, \alpha, \beta))$

**if**  $v \leq \alpha$  **then return**  $v$

$\beta \leftarrow \text{MIN}(\beta, v)$

**return**  $v$

# Algoritmo Poda $\alpha$ - $\beta$ (Práct)

```
function alphabeta(nodo, profundidad,  $\alpha$ ,  $\beta$ , deboMaximizar)
  if profundidad = 0 or nodo es terminal
    return evaluación heurística del nodo
  if deboMaximizar
    for each siguiente acción posible
       $\alpha := \max(\alpha, \text{alphabeta}(\text{hijo}, \text{profundidad} - 1, \alpha, \beta, \text{False}))$ 
      if  $\beta \leq \alpha$ 
        break
    return  $\alpha$ 
  else
    for each siguiente acción posible
       $\beta := \min(\beta, \text{alphabeta}(\text{hijo}, \text{profundidad} - 1, \alpha, \beta, \text{True}))$ 
      if  $\beta \leq \alpha$ 
        break
    return  $\beta$ 

(* Llamada inicial *)
alphabeta(origen, profundidad,  $-\infty$ ,  $+\infty$ , True)
```

## expectiminimax (T) /Expecitmax(P)

EXPECTIMINIMAX( $n$ ) =

$$\begin{cases} \text{UTILITY}(n) & \text{if } n \text{ is a terminal state} \\ \max_{s \in \text{Successors}(n)} \text{EXPECTIMINIMAX}(s) & \text{if } n \text{ is a MAX node} \\ \min_{s \in \text{Successors}(n)} \text{EXPECTIMINIMAX}(s) & \text{if } n \text{ is a MIN node} \\ \sum_{s \in \text{Successors}(n)} P(s) \cdot \text{EXPECTIMINIMAX}(s) & \text{if } n \text{ is a chance node} \end{cases}$$

```
function expectimax(nodo, profundidad, deboMaximizar)
  if profundidad = 0 or nodo es terminal
    return evaluación heurística del nodo
  if deboMaximizar
     $\alpha := -\infty$ 
    for each siguiente acción posible
       $\alpha := \max(\alpha, \text{expectimax}(\text{hijo}, \text{profundidad} - 1, \text{False}))$ 
    return  $\alpha$ 
  else
     $\beta := 0$ 
    numAcciones = 0
    for each siguiente acción posible
      numAcciones += 1
       $\beta := \beta + \text{expectimax}(\text{hijo}, \text{profundidad} - 1, \text{True})$ 
    return  $\beta / \text{numAcciones}$ 
```