

## Programación III

Ricardo Wehbe

UADE

3 de noviembre de 2021

## 1 Repaso de la clase anterior

## 2 Juegos

- El algoritmo Min-Max
- El juego de los corchos
- Back to Classics. El juego del ta-te-ti
- La poda alfa-beta

## 3 Ejercicios propuestos

## 1 Repaso de la clase anterior

## 2 Juegos

- El algoritmo Min-Max
- El juego de los corchos
- Back to Classics. El juego del ta-te-ti
- La poda alfa-beta

## 3 Ejercicios propuestos

# Búsqueda en grafos

# Búsqueda en grafos

- Para el recorrido de todos los nodos de un grafo existen dos criterios de búsqueda.

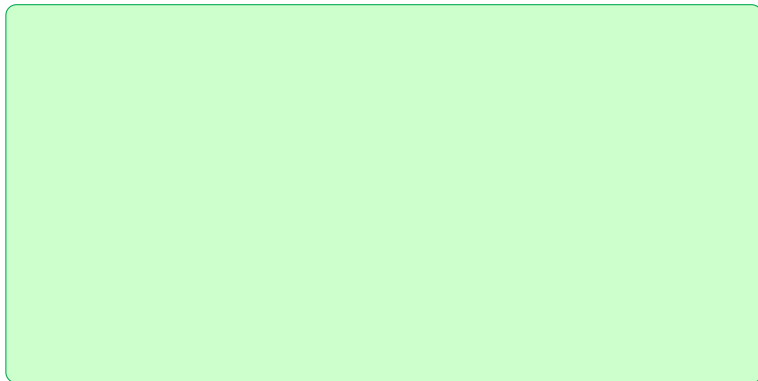
## Búsqueda en grafos

- Para el recorrido de todos los nodos de un grafo existen dos criterios de búsqueda.
- La *búsqueda en amplitud* (*breadth-first search*, o BFS): comienza la búsqueda por un nodo, toma luego sus adyacentes, visita cada uno de ellos y luego los adyacentes de los adyacentes.

## Búsqueda en grafos

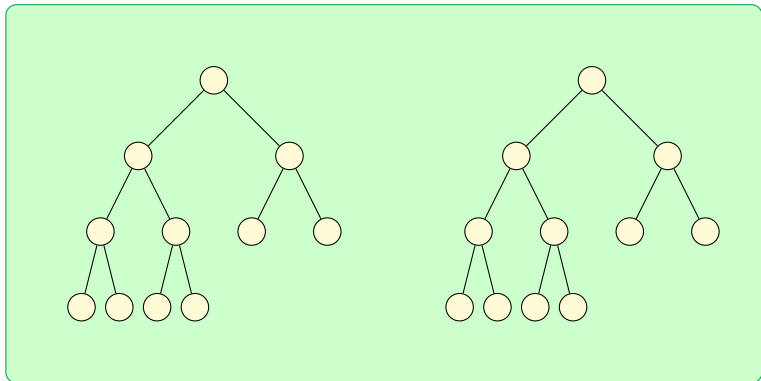
- Para el recorrido de todos los nodos de un grafo existen dos criterios de búsqueda.
- La *búsqueda en amplitud* (*breadth-first search*, o BFS): comienza la búsqueda por un nodo, toma luego sus adyacentes, visita cada uno de ellos y luego los adyacentes de los adyacentes.
- La *búsqueda en profundidad* (*depth-first search*, o DFS): comienza la búsqueda por un nodo, toma luego sus adyacentes y a partir de uno de ellos realiza a su vez la búsqueda en profundidad. Cuando termina la búsqueda para ese nodo adyacente continúa con el siguiente.

# Búsqueda en grafos: DFS y BFS

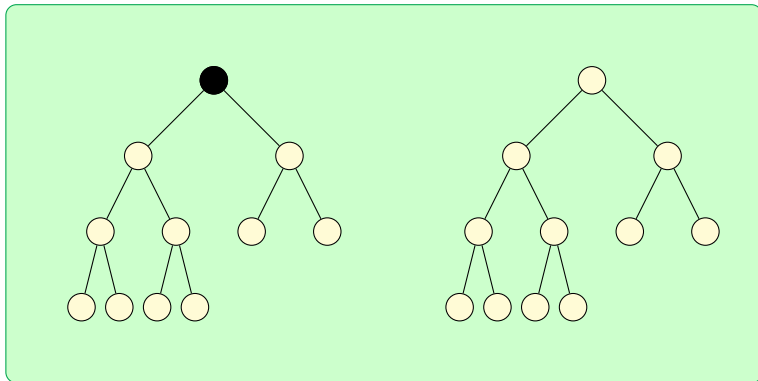




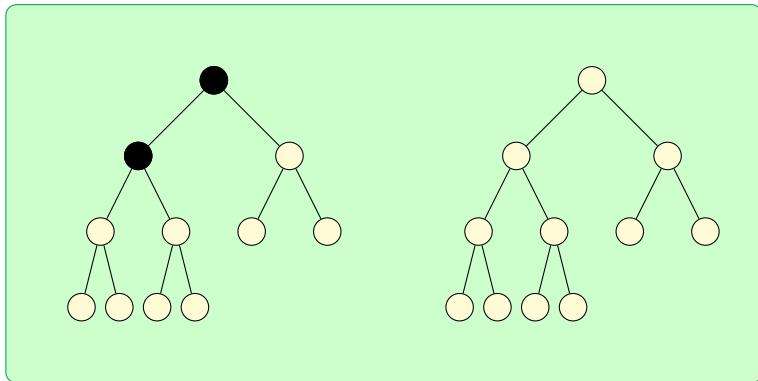
## Búsqueda en grafos: DFS y BFS



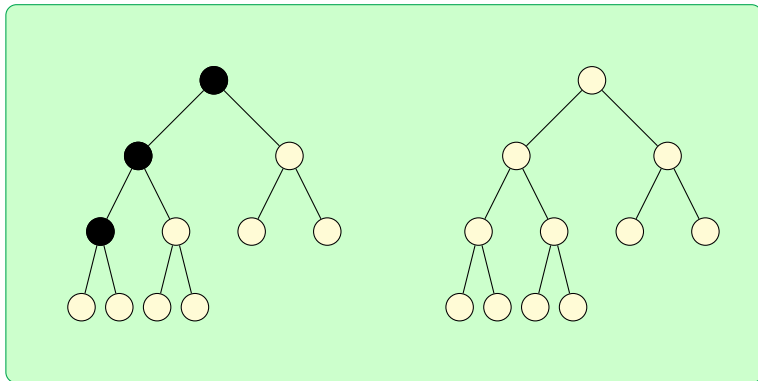
# Búsqueda en grafos: DFS y BFS



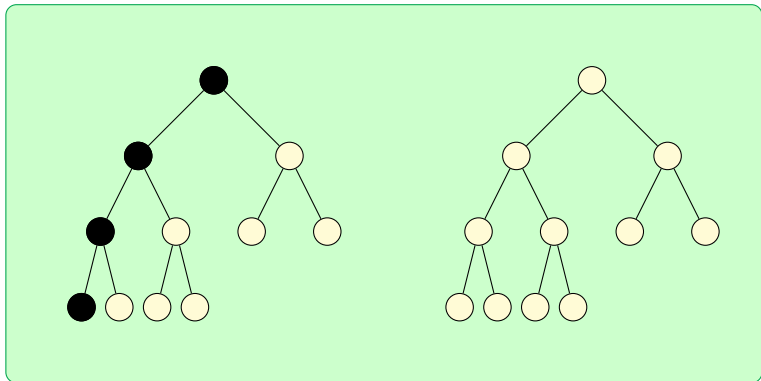
## Búsqueda en grafos: DFS y BFS



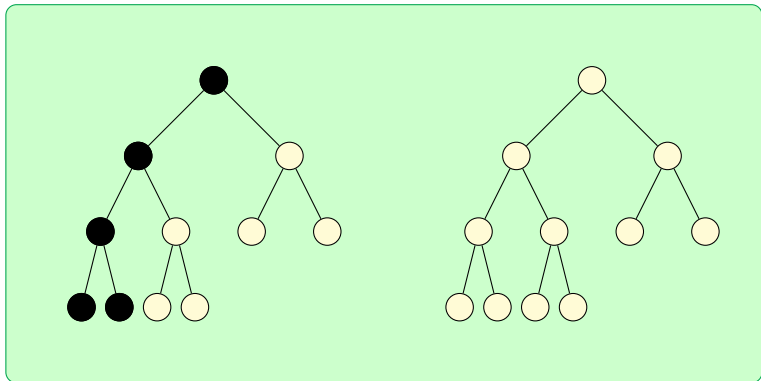
# Búsqueda en grafos: DFS y BFS



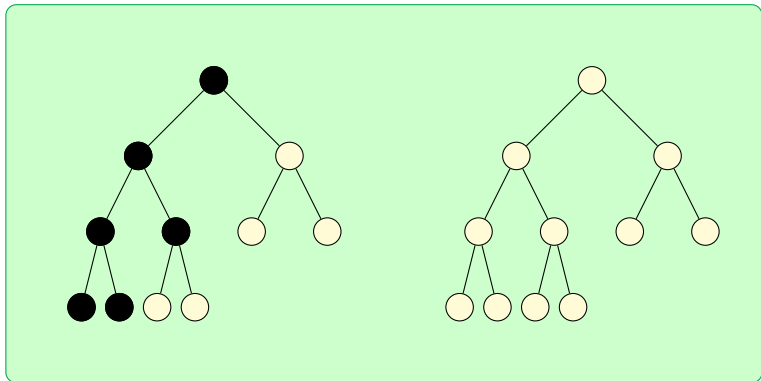
## Búsqueda en grafos: DFS y BFS



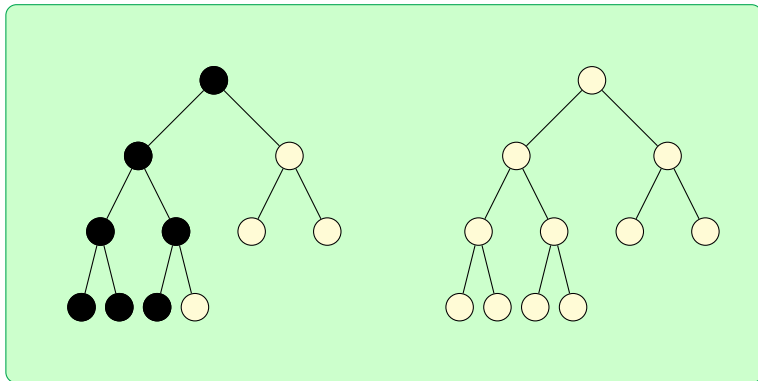
## Búsqueda en grafos: DFS y BFS



# Búsqueda en grafos: DFS y BFS

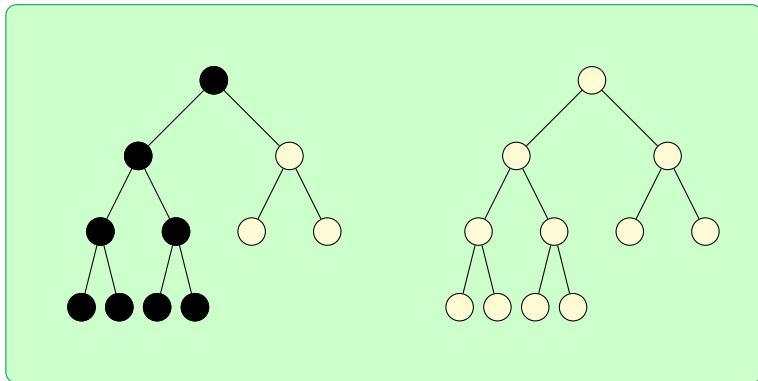


## Búsqueda en grafos: DFS y BFS

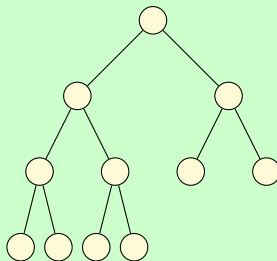
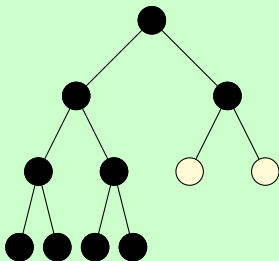




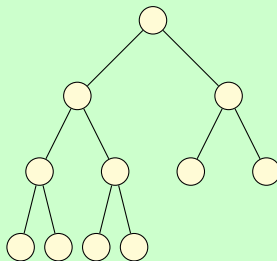
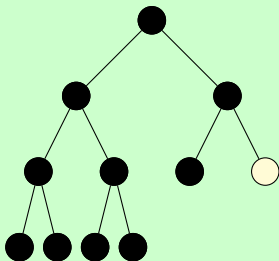
## Búsqueda en grafos: DFS y BFS



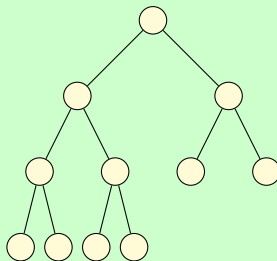
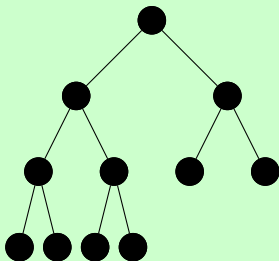
# Búsqueda en grafos: DFS y BFS



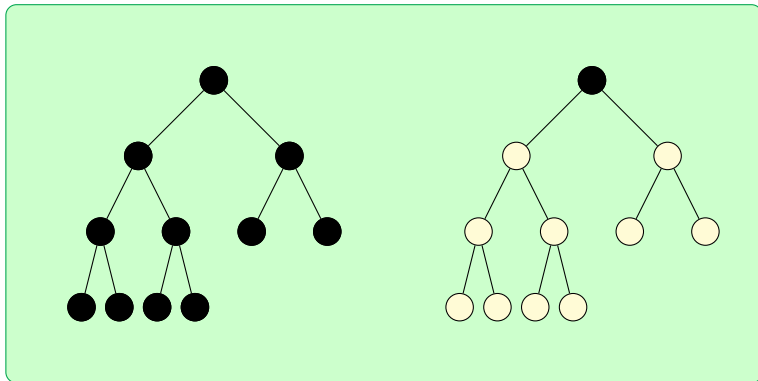
# Búsqueda en grafos: DFS y BFS



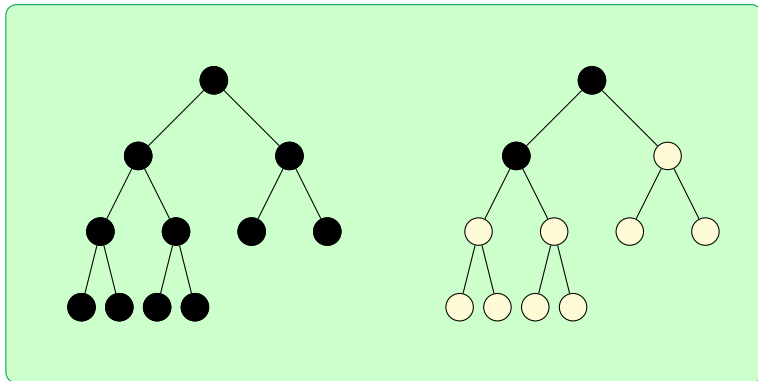
# Búsqueda en grafos: DFS y BFS



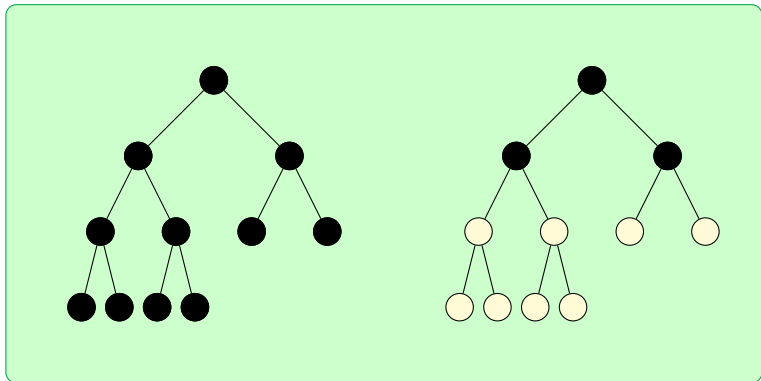
# Búsqueda en grafos: DFS y BFS



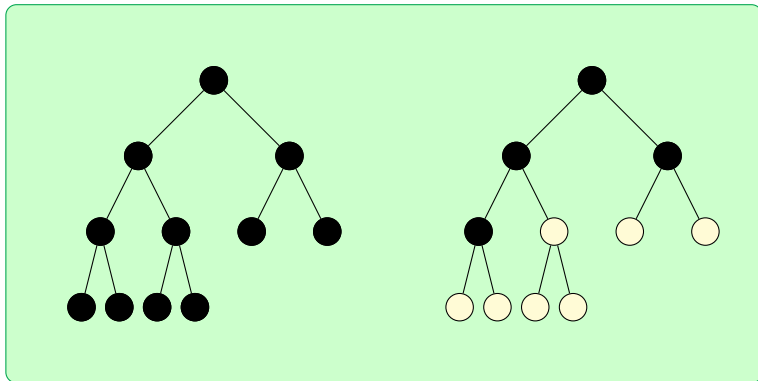
# Búsqueda en grafos: DFS y BFS



## Búsqueda en grafos: DFS y BFS

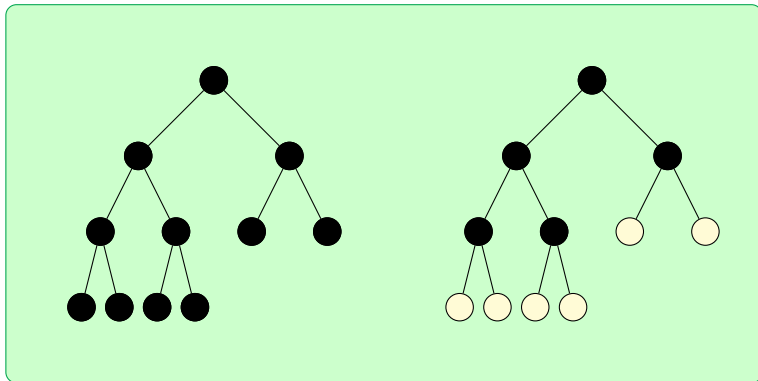


## Búsqueda en grafos: DFS y BFS

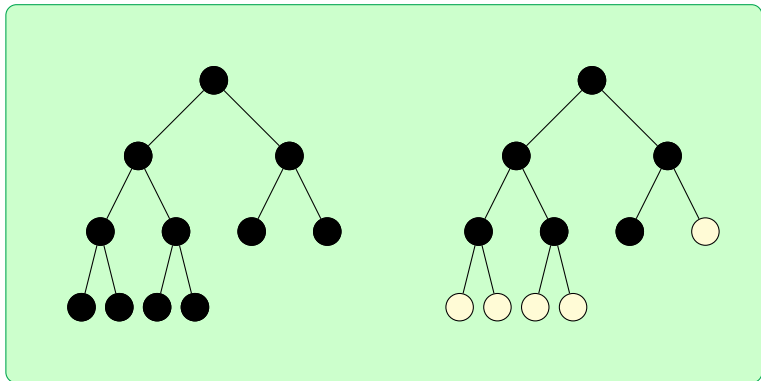




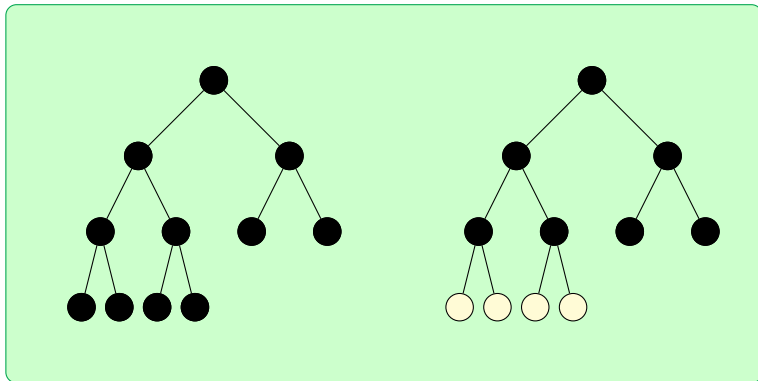
## Búsqueda en grafos: DFS y BFS



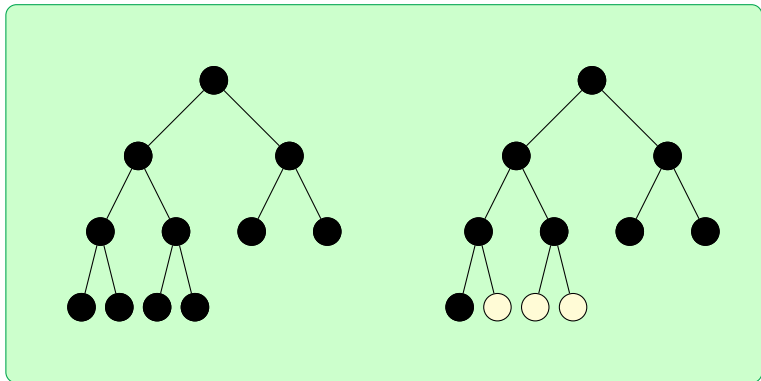
## Búsqueda en grafos: DFS y BFS



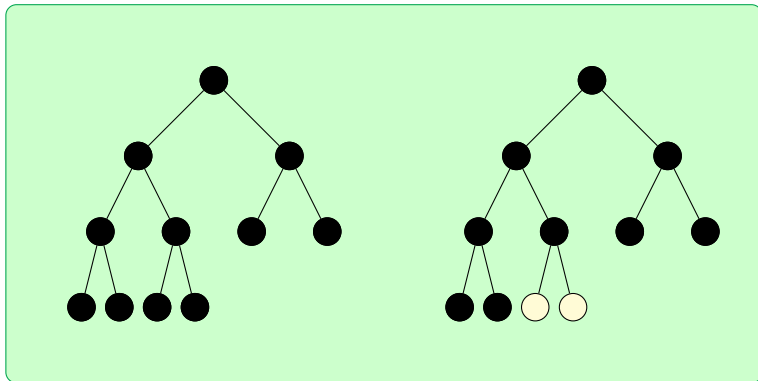
## Búsqueda en grafos: DFS y BFS



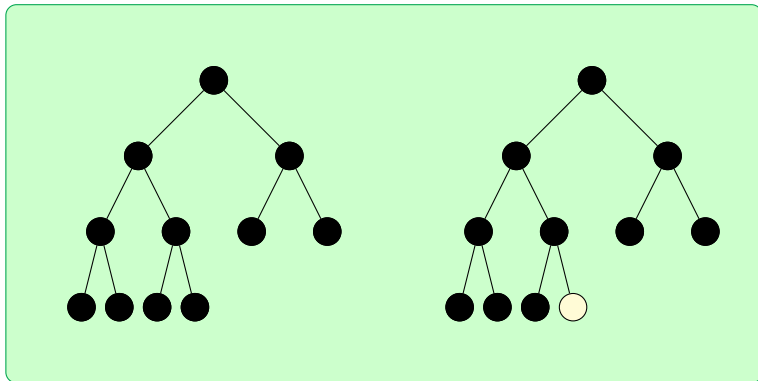
## Búsqueda en grafos: DFS y BFS



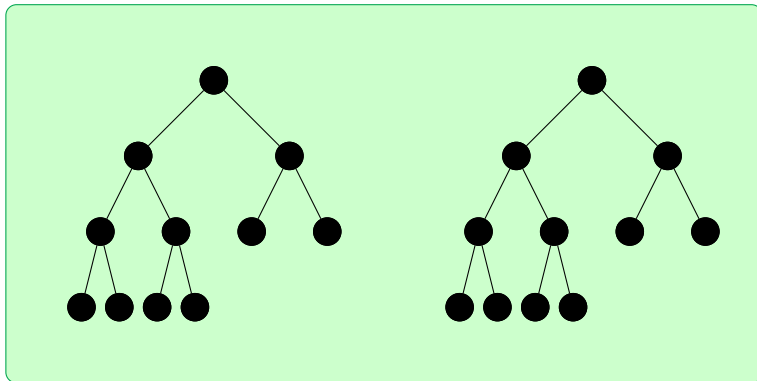
# Búsqueda en grafos: DFS y BFS



## Búsqueda en grafos: DFS y BFS



## Búsqueda en grafos: DFS y BFS



## 1 Repaso de la clase anterior

## 2 Juegos

- El algoritmo Min-Max
- El juego de los corchos
- Back to Classics. El juego del ta-te-ti
- La poda alfa-beta

## 3 Ejercicios propuestos



(...) le plaisir, délicieux et toujours nouveaux, d'une occupation inutile.

((...) el placer, delicioso y siempre nuevo, de una ocupación inútil.)

Henri de Régnier, *Les rencontres de M. de Bréot*

# Juegos

# Juegos

- Los juegos forman parte del paisaje cultural de cualquier civilización.

# Juegos

- Los juegos forman parte del paisaje cultural de cualquier civilización.
- Los juegos han despertado el interés de los economistas, pues permiten modelar situaciones en las que interactúan agentes autónomos.

# Juegos

- Los juegos forman parte del paisaje cultural de cualquier civilización.
- Los juegos han despertado el interés de los economistas, pues permiten modelar situaciones en las que interactúan agentes autónomos.
- Los juegos pueden ser colaborativos (todos los jugadores tienen un objetivo común) o adversariales (los jugadores tienen objetivos mutuamente excluyentes.)

# Juegos

- Los juegos forman parte del paisaje cultural de cualquier civilización.
- Los juegos han despertado el interés de los economistas, pues permiten modelar situaciones en las que interactúan agentes autónomos.
- Los juegos pueden ser colaborativos (todos los jugadores tienen un objetivo común) o adversariales (los jugadores tienen objetivos mutuamente excluyentes.)
- Aquí nos ocuparemos de juegos adversariales.

# Juegos y árboles

# Juegos y árboles

- Muchos juegos tienen un número finito de estados (ajedrez, damas, ta-te-ti, nim, cuatro-en-fila, &c.) Estos juegos pueden ser modelados por árboles que permiten la aplicación de *backtracking*.



# Juegos y árboles

- Muchos juegos tienen un número finito de estados (ajedrez, damas, ta-te-ti, nim, cuatro-en-fila, &c.) Estos juegos pueden ser modelados por árboles que permiten la aplicación de *backtracking*.
- Cada nivel del árbol corresponde al turno de un jugador.

# Juegos y árboles

- Muchos juegos tienen un número finito de estados (ajedrez, damas, ta-te-ti, nim, cuatro-en-fila, &c.) Estos juegos pueden ser modelados por árboles que permiten la aplicación de *backtracking*.
- Cada nivel del árbol corresponde al turno de un jugador.
- Adumiremos la existencia de dos jugadores hipotéticos que llamaremos *Max* y *Min*, que harán una movida por turno. *Max* será el pimer jugador.

# Juegos y árboles

- Muchos juegos tienen un número finito de estados (ajedrez, damas, ta-te-ti, nim, cuatro-en-fila, &c.) Estos juegos pueden ser modelados por árboles que permiten la aplicación de *backtracking*.
- Cada nivel del árbol corresponde al turno de un jugador.
- Adumiremos la existencia de dos jugadores hipotéticos que llamaremos *Max* y *Min*, que harán una movida por turno. *Max* será el pimer jugador.
- En otras palabras, en la raíz del árbol (nivel 0) juega *Max*; en el nivel 1 responde *Min*; en el nivel 2 responde *Max*, y así ducesivamente.

# Juegos y árboles

- Muchos juegos tienen un número finito de estados (ajedrez, damas, ta-te-ti, nim, cuatro-en-fila, &c.) Estos juegos pueden ser modelados por árboles que permiten la aplicación de *backtracking*.
- Cada nivel del árbol corresponde al turno de un jugador.
- Adumiremos la existencia de dos jugadores hipotéticos que llamaremos *Max* y *Min*, que harán una movida por turno. *Max* será el pimer jugador.
- En otras palabras, en la raíz del árbol (nivel 0) juega *Max*; en el nivel 1 responde *Min*; en el nivel 2 responde *Max*, y así ducesivamente.
- Observe que no es siempre posible en la práctica construir el árbol completo de un juego.

# Juegos y árboles

- Muchos juegos tienen un número finito de estados (ajedrez, damas, ta-te-ti, nim, cuatro-en-fila, &c.) Estos juegos pueden ser modelados por árboles que permiten la aplicación de *backtracking*.
- Cada nivel del árbol corresponde al turno de un jugador.
- Adumiremos la existencia de dos jugadores hipotéticos que llamaremos *Max* y *Min*, que harán una movida por turno. *Max* será el pimer jugador.
- En otras palabras, en la raíz del árbol (nivel 0) juega *Max*; en el nivel 1 responde *Min*; en el nivel 2 responde *Max*, y así ducesivamente.
- Observe que no es siempre posible en la práctica construir el árbol completo de un juego.
- En resumen, el árbol de un juego es un árbol dirigido cuyos vértices representan un estado del juego y cuyas aristas representan movidas.

# Estrategias ganadoras y nodos $N$ y $P$

## Estrategias ganadoras y nodos $N$ y $P$

- Una *estrategia ganadora* es un plan para jugar un juego de manera que un jugador gana independientemente de lo que su oponente haga.

## Estrategias ganadoras y nodos $N$ y $P$

- Una *estrategia ganadora* es un plan para jugar un juego de manera que un jugador gana independientemente de lo que su oponente haga.
- Un nodo  $N$  es un nodo en el que el jugador que tiene el turno de jugar tiene una estrategia ganadora. El nombre viene de que el jugador ganador hace la siguiente (*next*) jugada.



## Estrategias ganadoras y nodos $N$ y $P$

- Una *estrategia ganadora* es un plan para jugar un juego de manera que un jugador gana independientemente de lo que su oponente haga.
- Un nodo  $N$  es un nodo en el que el jugador que tiene el turno de jugar tiene una estrategia ganadora. El nombre viene de que el jugador ganador hace la siguiente (*next*) jugada.
- Un nodo  $P$  es un nodo en el que el jugador que tiene el turno de jugar no puede evitar la derrota si su rival juega correctamente. El nombre viene de que el jugador ganador hizo la jugada anterior (*previous*.)

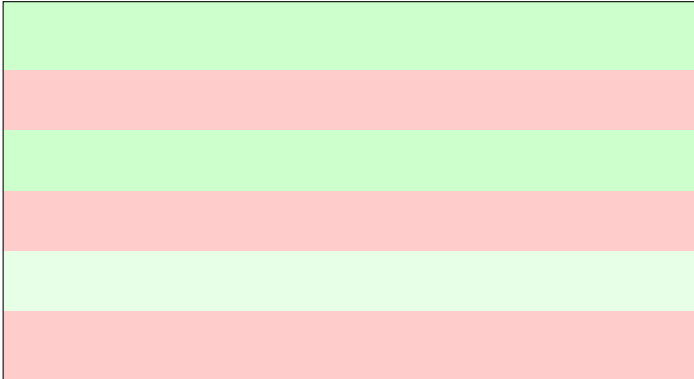
## Estrategias ganadoras y nodos $N$ y $P$

- Una *estrategia ganadora* es un plan para jugar un juego de manera que un jugador gana independientemente de lo que su oponente haga.
- Un nodo  $N$  es un nodo en el que el jugador que tiene el turno de jugar tiene una estrategia ganadora. El nombre viene de que el jugador ganador hace la siguiente (*next*) jugada.
- Un nodo  $P$  es un nodo en el que el jugador que tiene el turno de jugar no puede evitar la derrota si su rival juega correctamente. El nombre viene de que el jugador ganador hizo la jugada anterior (*previous*.)
- Un nodo puede no ser  $N$  ni  $P$ .

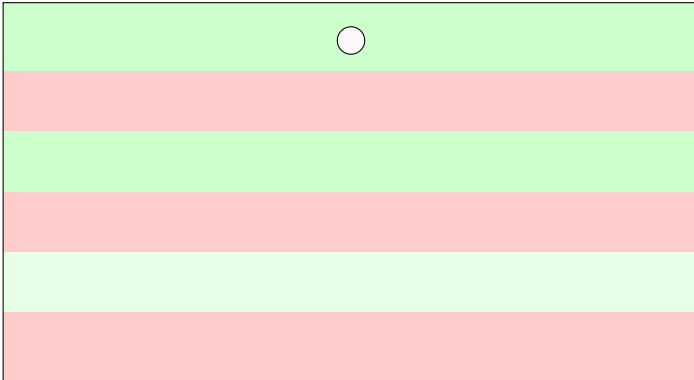
## Estrategias ganadoras y nodos $N$ y $P$

- Una *estrategia ganadora* es un plan para jugar un juego de manera que un jugador gana independientemente de lo que su oponente haga.
- Un nodo  $N$  es un nodo en el que el jugador que tiene el turno de jugar tiene una estrategia ganadora. El nombre viene de que el jugador ganador hace la siguiente (*next*) jugada.
- Un nodo  $P$  es un nodo en el que el jugador que tiene el turno de jugar no puede evitar la derrota si su rival juega correctamente. El nombre viene de que el jugador ganador hizo la jugada anterior (*previous*.)
- Un nodo puede no ser  $N$  ni  $P$ .
- Si todos los hijos de un nodo son nodos  $N$ , entonces se trata de un nodo  $P$ ; si alguno de los hijos de un nodo es un nodo  $P$ , entonces se trata de un nodo  $N$ .
- Un ejemplo se muestra en la siguiente diapositiva.

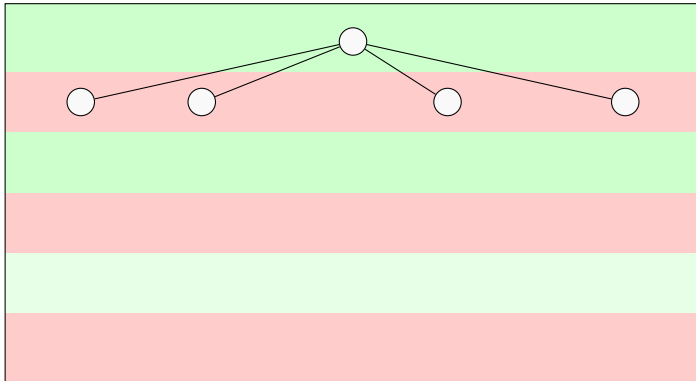
# Un primer ejemplo



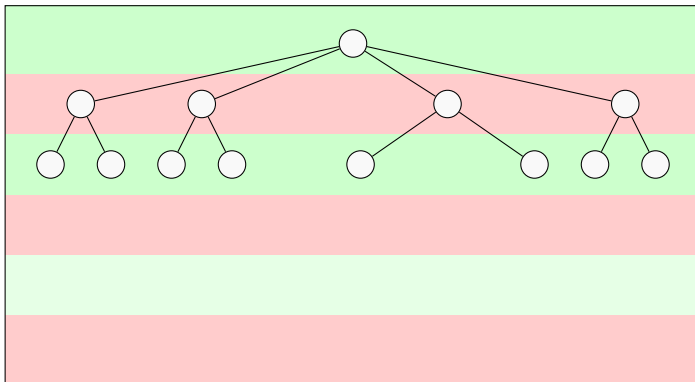
# Un primer ejemplo



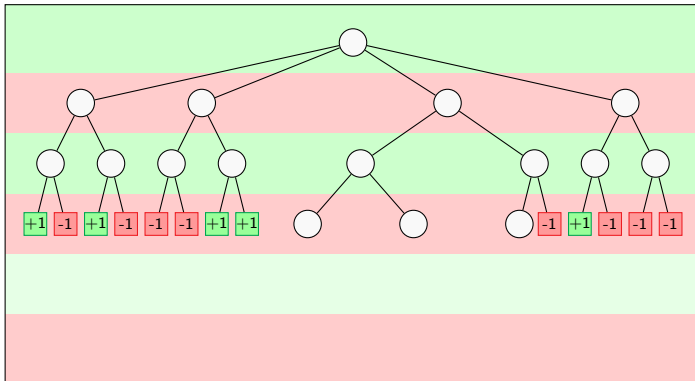
## Un primer ejemplo



## Un primer ejemplo

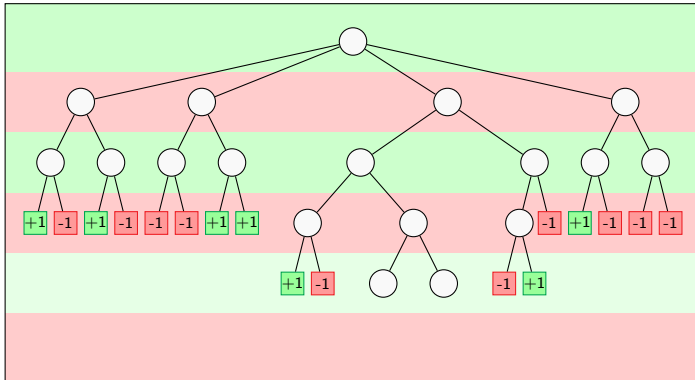


## Un primer ejemplo

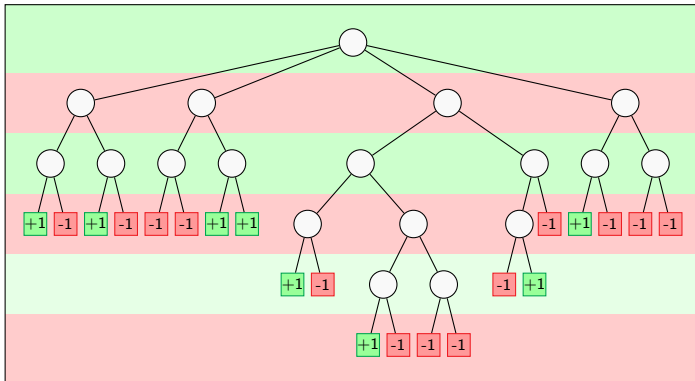




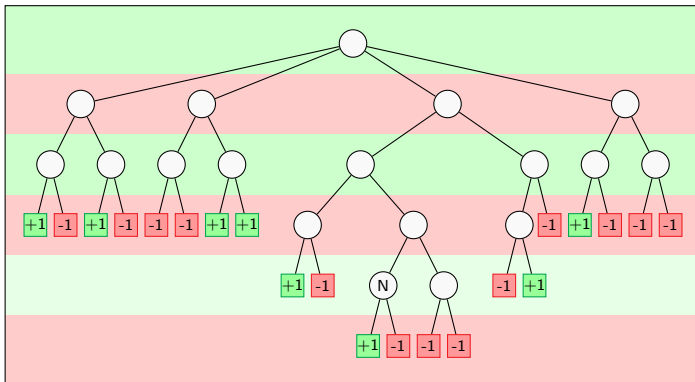
## Un primer ejemplo



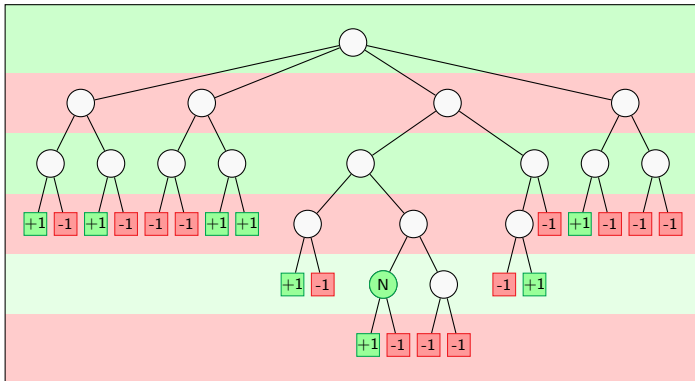
## Un primer ejemplo



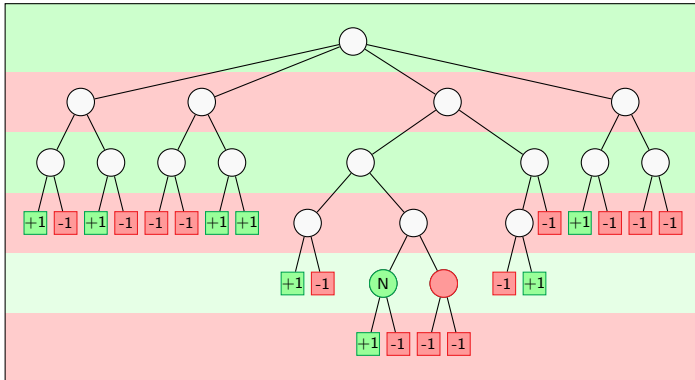
## Un primer ejemplo



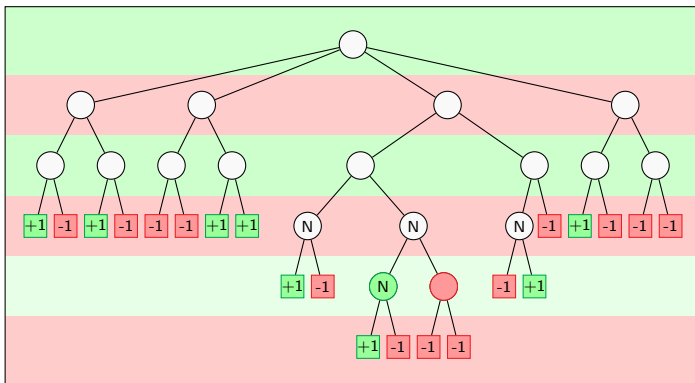
# Un primer ejemplo



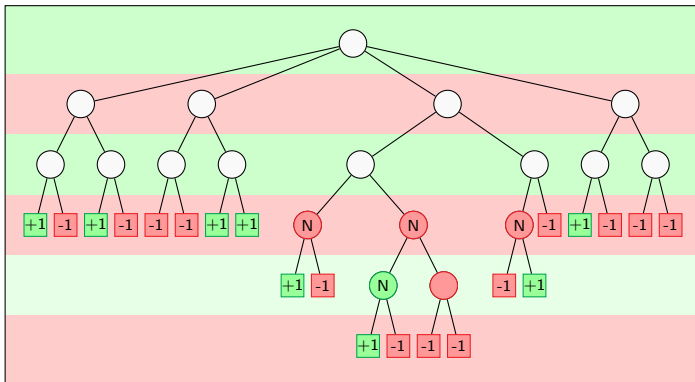
# Un primer ejemplo



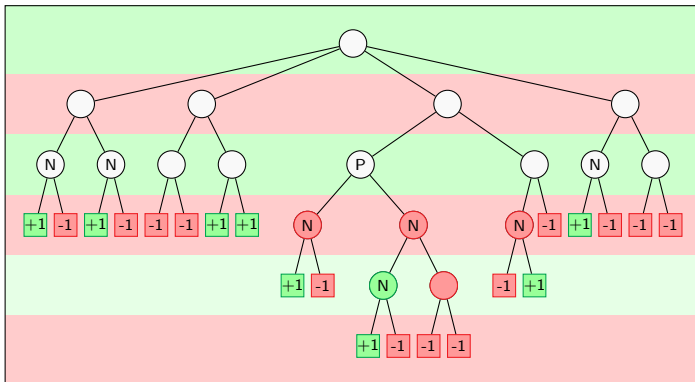
# Un primer ejemplo



## Un primer ejemplo

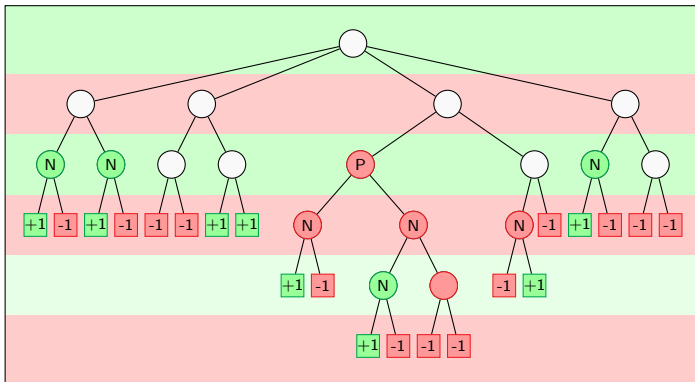


## Un primer ejemplo

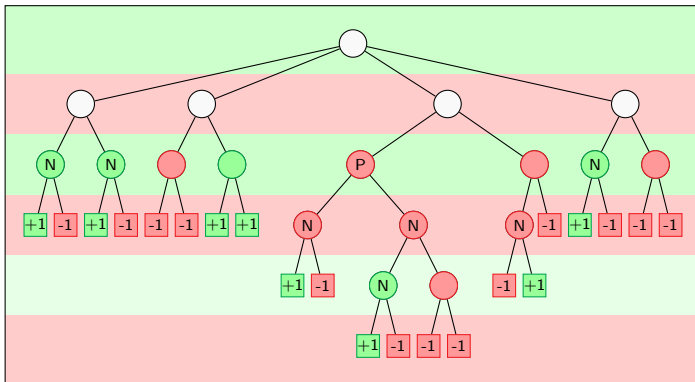




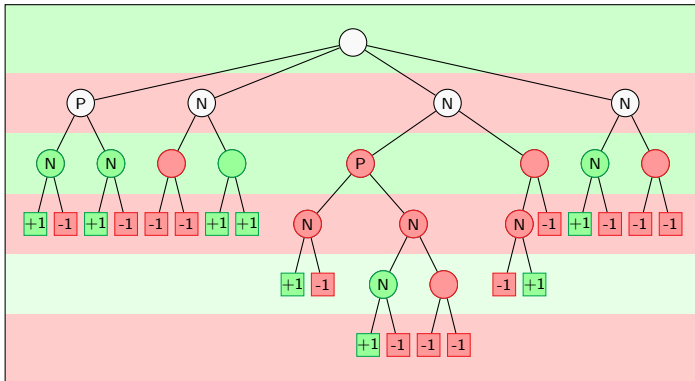
## Un primer ejemplo



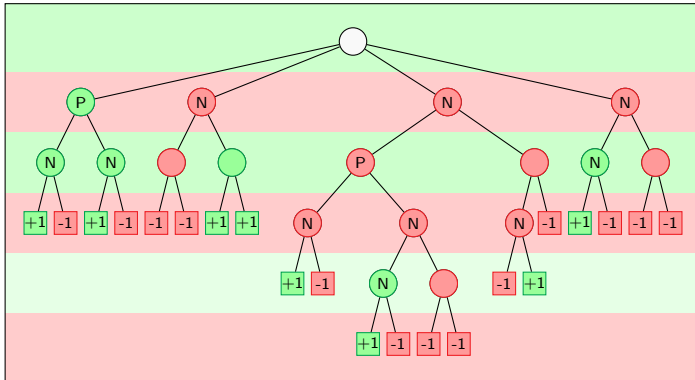
## Un primer ejemplo



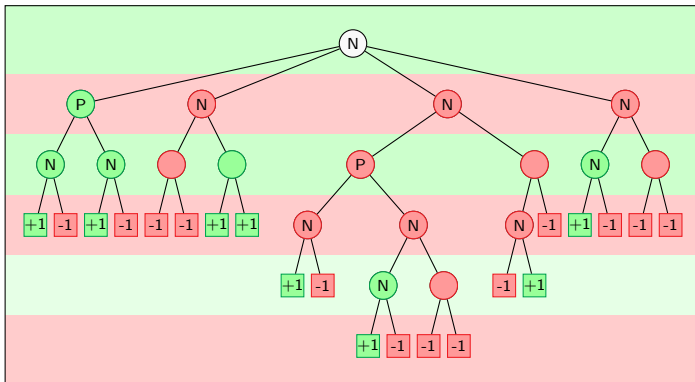
## Un primer ejemplo



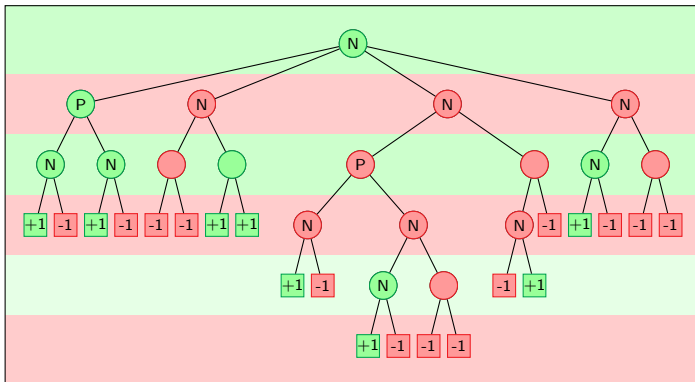
## Un primer ejemplo



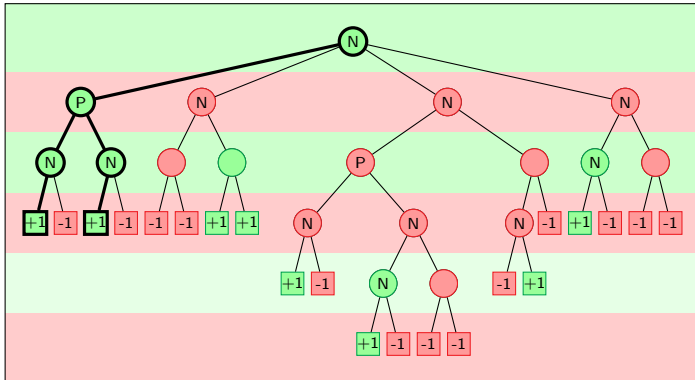
## Un primer ejemplo



## Un primer ejemplo



## Un primer ejemplo



## 1 Repaso de la clase anterior

## 2 Juegos

- El algoritmo Min-Max
- El juego de los corchos
- Back to Classics. El juego del ta-te-ti
- La poda alfa-beta

## 3 Ejercicios propuestos



# El algoritmo Min-Max

# El algoritmo Min-Max

- Volvamos con nuestros amigos, *Max* y *Min*.

# El algoritmo Min-Max

- Volvamos con nuestros amigos, *Max* y *Min*.
- *Max* quiere maximizar el resultado del juego; *Min* quiere minimizarlo.

# El algoritmo Min-Max

- Volvamos con nuestros amigos, *Max* y *Min*.
- *Max* quiere maximizar el resultado del juego; *Min* quiere minimizarlo.
- En la versión más simple, sólo hay dos resultados:  $+1$  (*Max* gana) y  $-1$  (*Min* gana.)

# El algoritmo Min-Max

- Volvamos con nuestros amigos, *Max* y *Min*.
- *Max* quiere maximizar el resultado del juego; *Min* quiere minimizarlo.
- En la versión más simple, sólo hay dos resultados:  $+1$  (*Max* gana) y  $-1$  (*Min* gana.)
- Algunos juegos pueden resultar en empates; en este caso, el resultado es,  $0$ .

# El algoritmo Min-Max

- Volvamos con nuestros amigos, *Max* y *Min*.
- *Max* quiere maximizar el resultado del juego; *Min* quiere minimizarlo.
- En la versión más simple, sólo hay dos resultados:  $+1$  (*Max* gana) y  $-1$  (*Min* gana.)
- Algunos juegos pueden resultar en empates; en este caso, el resultado es,  $0$ .
- En otras versiones, el resultado es un número; por ejemplo, en un juego de cartas en el que ambos bandos acumulan puntos en varias manos.

# El algoritmo Min-Max

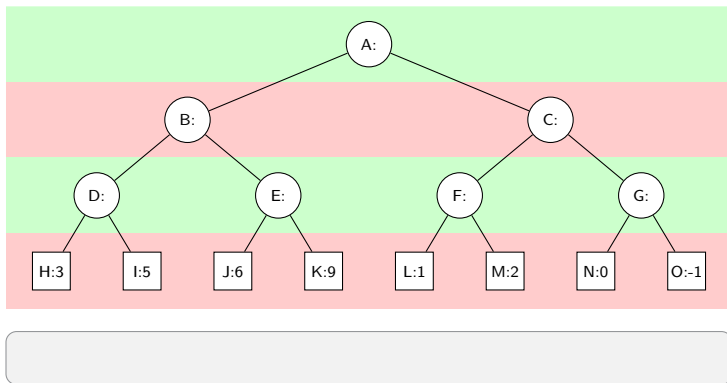
- Volvamos con nuestros amigos, *Max* y *Min*.
- *Max* quiere maximizar el resultado del juego; *Min* quiere minimizarlo.
- En la versión más simple, sólo hay dos resultados: *+1* (*Max* gana) y *-1* (*Min* gana.)
- Algunos juegos pueden resultar en empates; en este caso, el resultado es, *0*.
- En otras versiones, el resultado es un número; por ejemplo, en un juego de cartas en el que ambos bandos acumulan puntos en varias manos.
- A continuación, veremos el pseudo-código y después un ejemplo simple.

# The *MinMax* Algorithm for Games

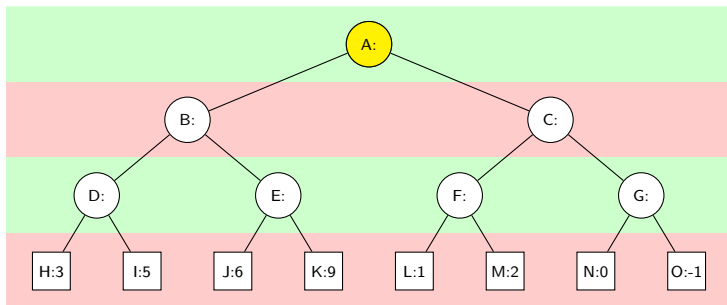
```
1.  algoritmo MinMax (estado e, nivel n)           // estado es un vértice; nivel es MAX o MIN
2.  if hoja(e, n) {
3.      return resultado(e, n)                     // devuelve un número
4.  } else {
5.      if (n == MAX) {
6.          valor =  $-\infty$                            // peor caso posible
7.      } else {
8.          valor =  $+\infty$                            // peor caso posible
9.      }
10.     foreach sig = hijo(e) do {                   // iteramos para todos los hijos
11.         if (n == Max) {
12.             valor = max(valor, MinMax(sig, MIN))
13.         } else {
14.             valor = min(valor, MinMax(sig, MAX))
15.         }
16.     }
17. }
18. return valor
```



## El algoritmo Min-Max. Un ejemplo

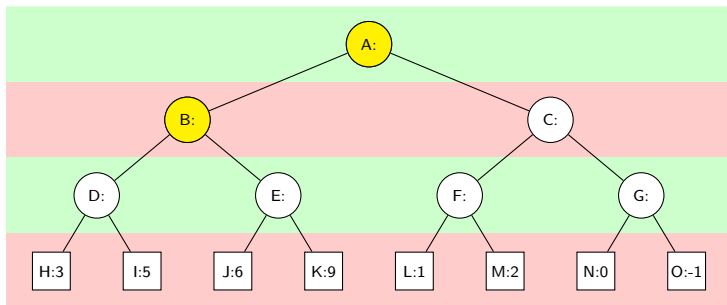


## El algoritmo Min-Max. Un ejemplo



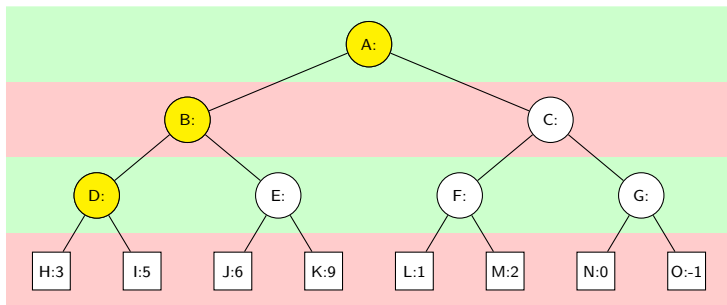
*Max* elige el máximo entre *B* y *C*; evalúa primero *B*.

## El algoritmo Min-Max. Un ejemplo



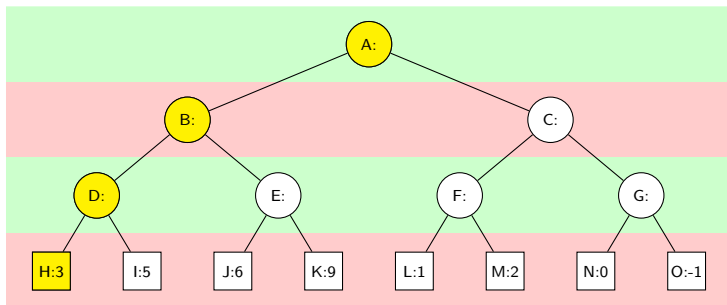
*Min* elige el mínimo entre *D* y *E*; evalúa primero *D*.

## El algoritmo Min-Max. Un ejemplo



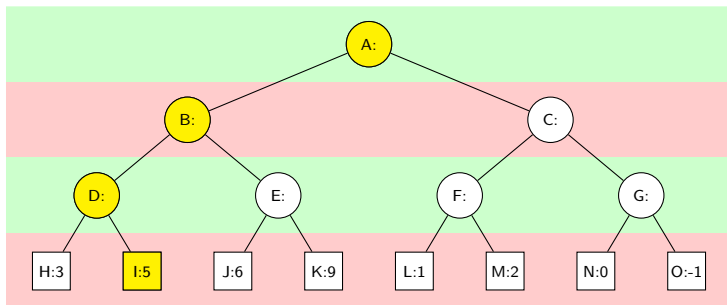
*Max* elige el máximo entre *H* y *I*; evalúa primero *H*.

## El algoritmo Min-Max. Un ejemplo



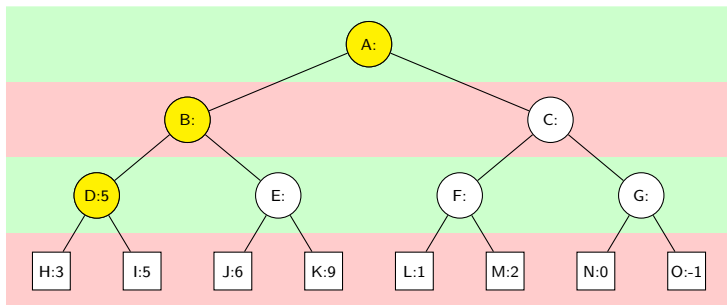
El nodo *H* devuelve 3.

## El algoritmo Min-Max. Un ejemplo



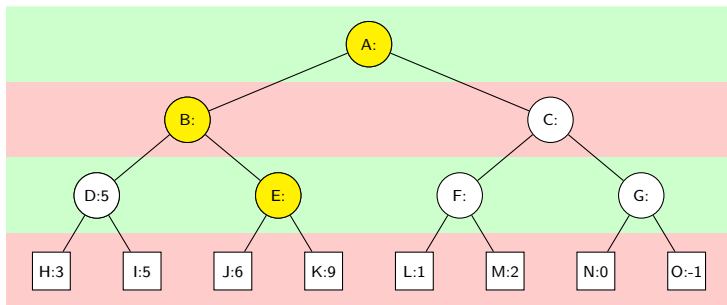
El nodo / devuelve 5.

## El algoritmo Min-Max. Un ejemplo



El nodo *D* devuelve  $\max(3, 5) = 5$ .

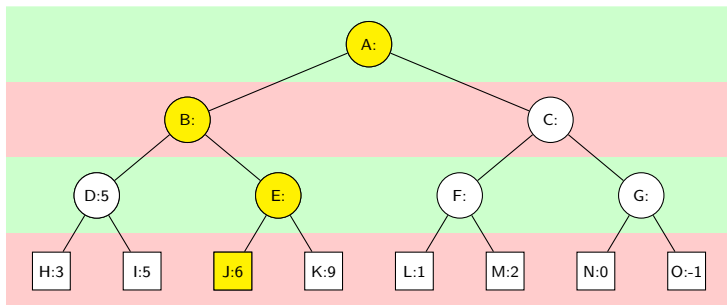
## El algoritmo Min-Max. Un ejemplo



*Max* elige el máximo entre *J* y *K*; evalúa primero *J*.

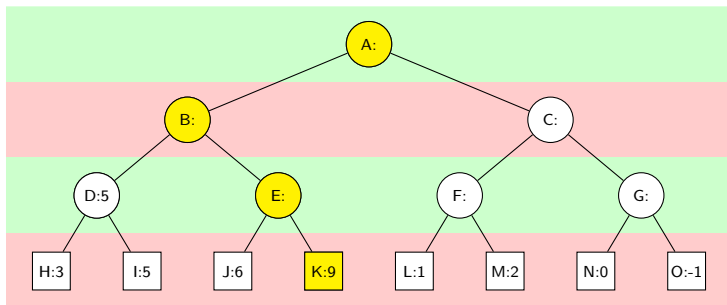


## El algoritmo Min-Max. Un ejemplo



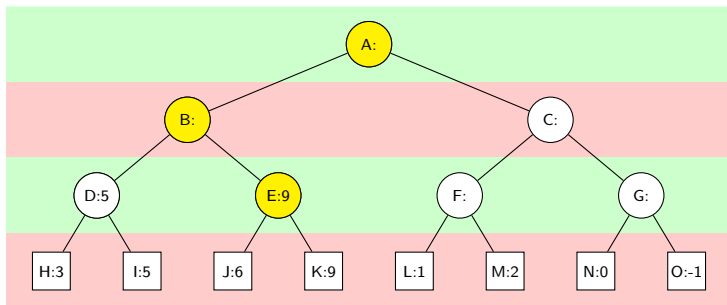
El nodo *J* devuelve 6.

## El algoritmo Min-Max. Un ejemplo



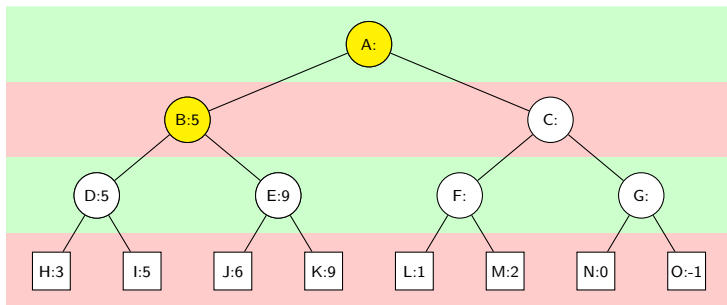
El nodo *K* devuelve 9.

## El algoritmo Min-Max. Un ejemplo



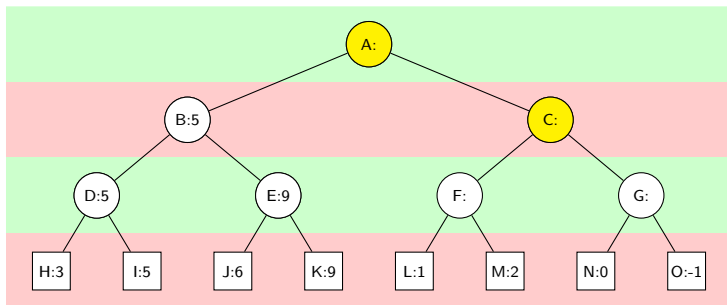
El nodo *E* devuelve  $\max(6, 9) = 9$ .

## El algoritmo Min-Max. Un ejemplo



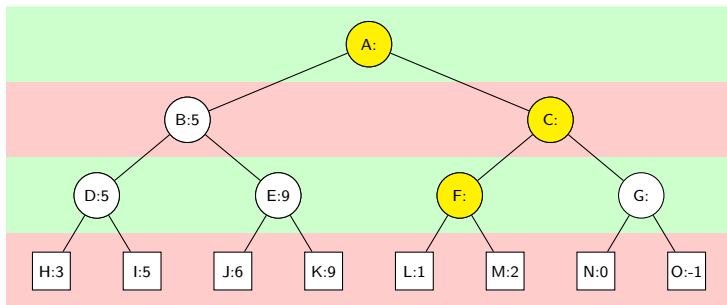
El nodo *B* devuelve  $\min(5, 9) = 5$ .

## El algoritmo Min-Max. Un ejemplo



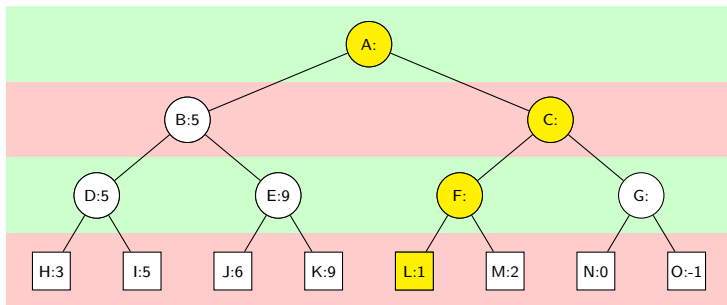
*Min*elige el mínimo entre  $F$  y  $G$ ; evalúa primero  $F$ .

## El algoritmo Min-Max. Un ejemplo



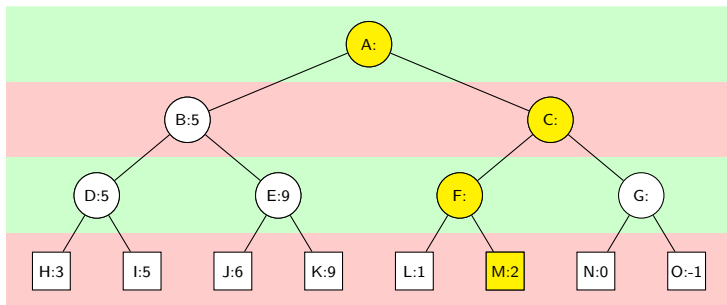
*Max* elige el máximo entre *L* y *M*; evalúa primero *L*.

## El algoritmo Min-Max. Un ejemplo



El nodo **L** devuelve 1.

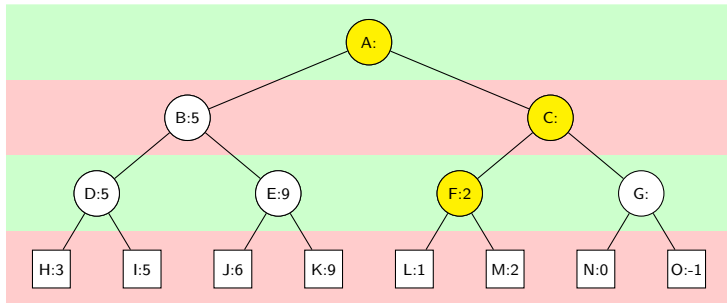
## El algoritmo Min-Max. Un ejemplo



El nodo *M* devuelve 2.

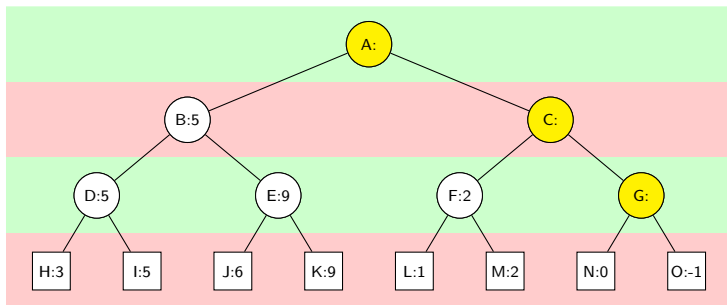


## El algoritmo Min-Max. Un ejemplo



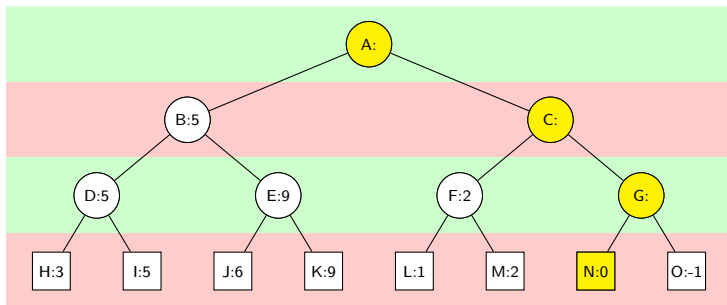
El nodo  $F$  devuelve  $\max(1, 2) = 2$ .

## El algoritmo Min-Max. Un ejemplo



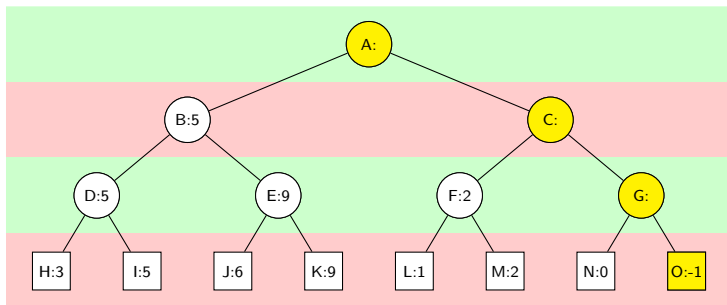
*Max* elige el máximo entre *N* y *O*; evalúa primero *N*.

## El algoritmo Min-Max. Un ejemplo



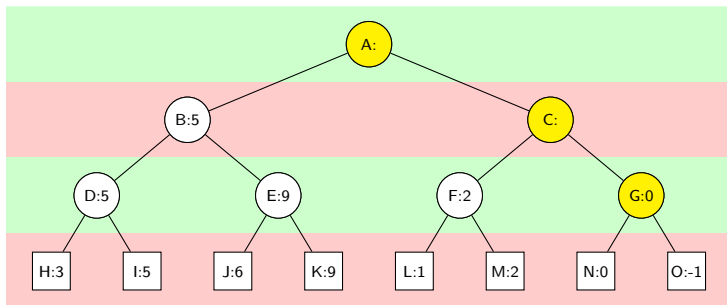
El nodo **N** devuelve 0.

## El algoritmo Min-Max. Un ejemplo



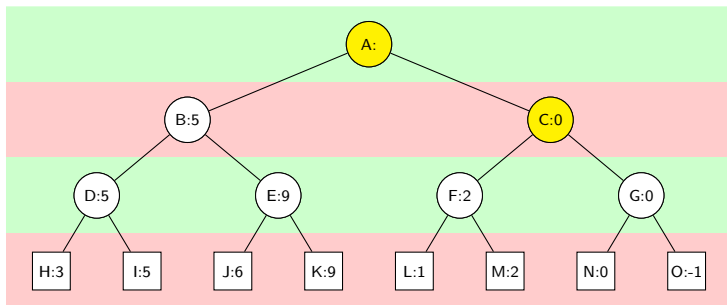
El nodo **O** devuelve **-1**.

## El algoritmo Min-Max. Un ejemplo



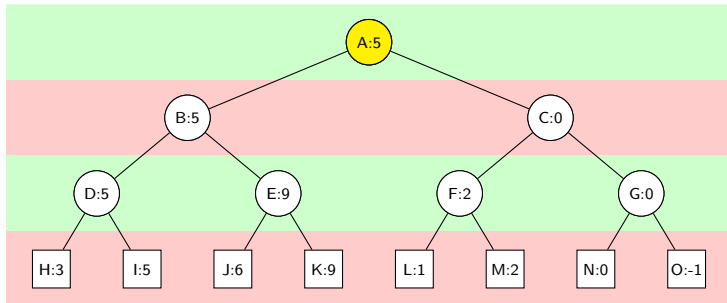
El nodo *F* devuelve  $\max(0, -1) = 0$ .

## El algoritmo Min-Max. Un ejemplo



El nodo **C** devuelve  $\min(2, 0) = 0$ .

## El algoritmo Min-Max. Un ejemplo



El nodo **A** devuelve  $\min(5, 0) = 5$ .

## 1 Repaso de la clase anterior

## 2 Juegos

- El algoritmo Min-Max
- El juego de los corchos
- Back to Classics. El juego del ta-te-ti
- La poda alfa-beta

## 3 Ejercicios propuestos



# Un ejemplo. El juego de los corchos

## Un ejemplo. El juego de los corchos

- Dos amigos resuelven jugar al siguiente juego con los once corchos de las botellas de *Château d'Yquem* (Sauternes, vintage 2009) que acaban de tomarse. Los once corchos están sobre la mesa y cada jugador debe retirar por turno uno, dos o tres corchos.

## Un ejemplo. El juego de los corchos

- Dos amigos resuelven jugar al siguiente juego con los once corchos de las botellas de *Château d'Yquem* (Sauternes, vintage 2009) que acaban de tomarse. Los once corchos están sobre la mesa y cada jugador debe retirar por turno uno, dos o tres corchos.
- Ningún jugador puede “pasar.”

## Un ejemplo. El juego de los corchos

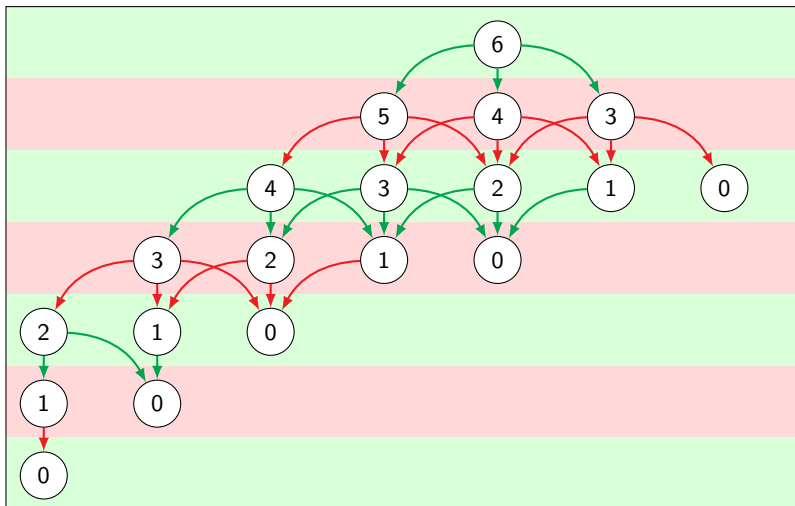
- Dos amigos resuelven jugar al siguiente juego con los once corchos de las botellas de *Château d'Yquem* (Sauternes, vintage 2009) que acaban de tomarse. Los once corchos están sobre la mesa y cada jugador debe retirar por turno uno, dos o tres corchos.
- Ningún jugador puede “pasar.”
- Pierde el jugador que retira el último corcho.

## Un ejemplo. El juego de los corchos

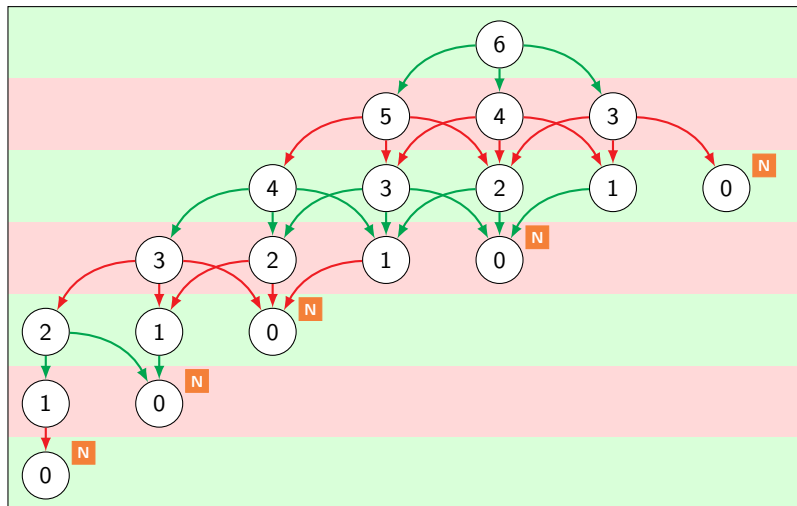
- Dos amigos resuelven jugar al siguiente juego con los once corchos de las botellas de *Château d'Yquem* (Sauternes, vintage 2009) que acaban de tomarse. Los once corchos están sobre la mesa y cada jugador debe retirar por turno uno, dos o tres corchos.
- Ningún jugador puede “pasar.”
- Pierde el jugador que retira el último corcho.
- El juego se representa por un árbol (en nuestro ejemplo, un grafo) cuyos vértices están etiquetados con el número actual de corchos que quedan.

# El juego de los corchos a partir del estado 6

## El juego de los corchos a partir del estado 6

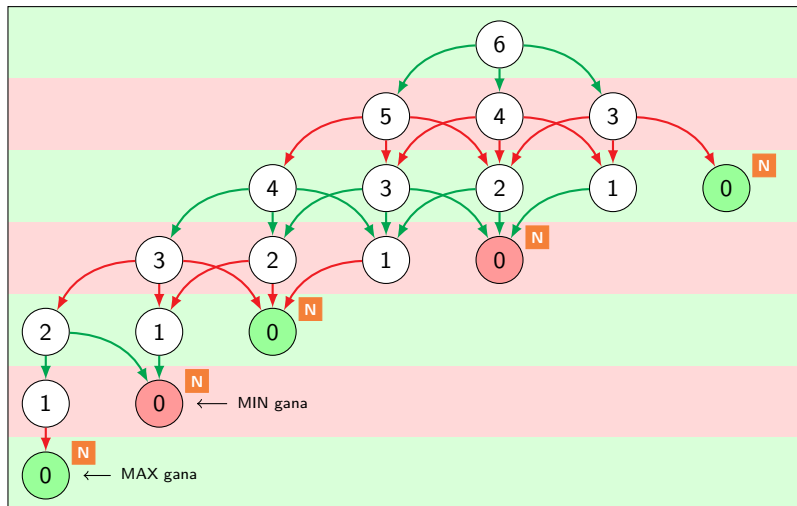


## El juego de los corchos a partir del estado 6

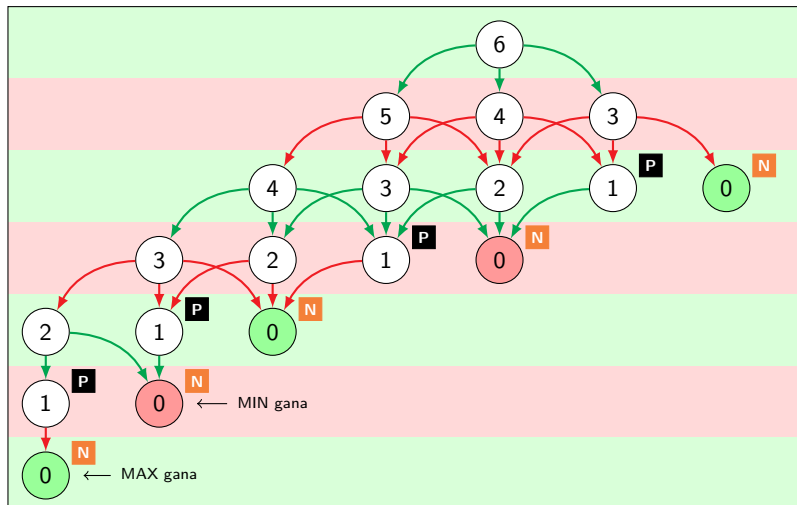




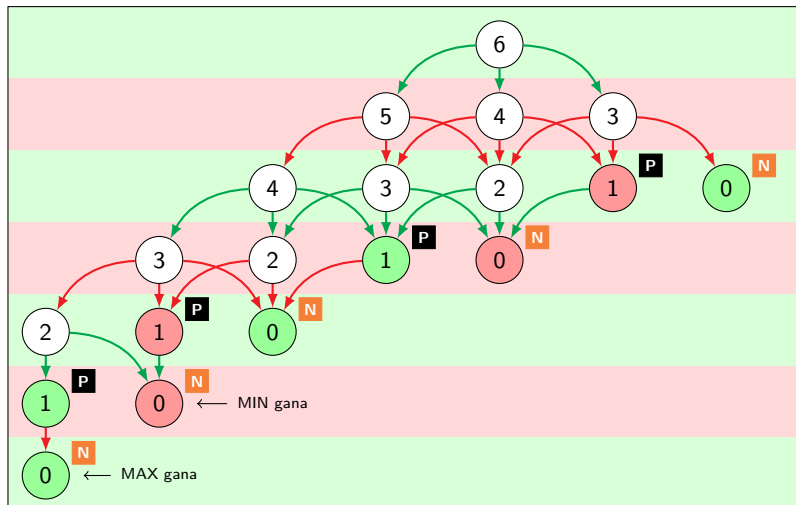
## El juego de los corchos a partir del estado 6



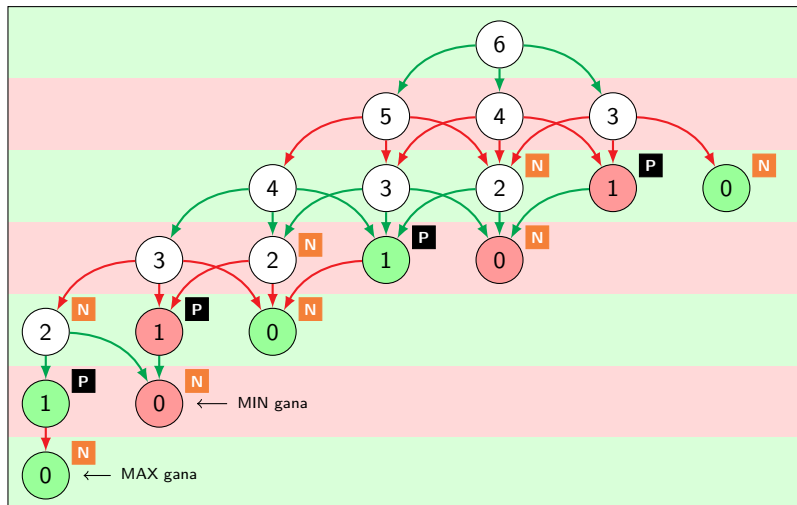
## El juego de los corchos a partir del estado 6



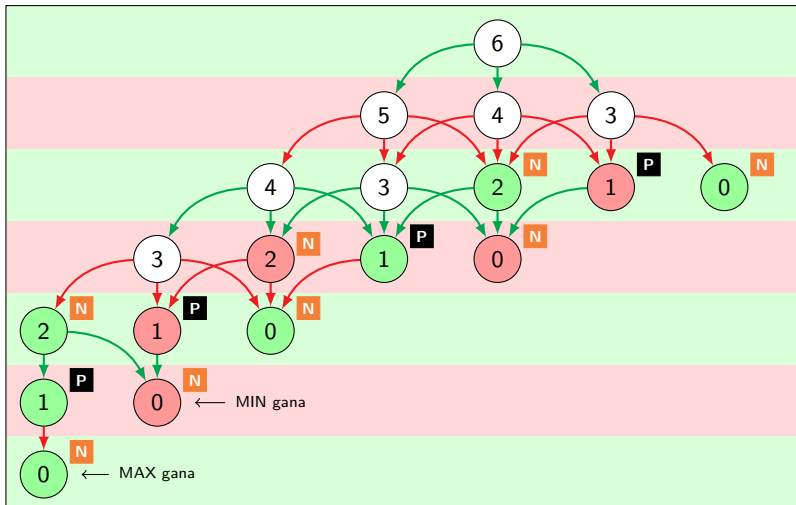
## El juego de los corchos a partir del estado 6



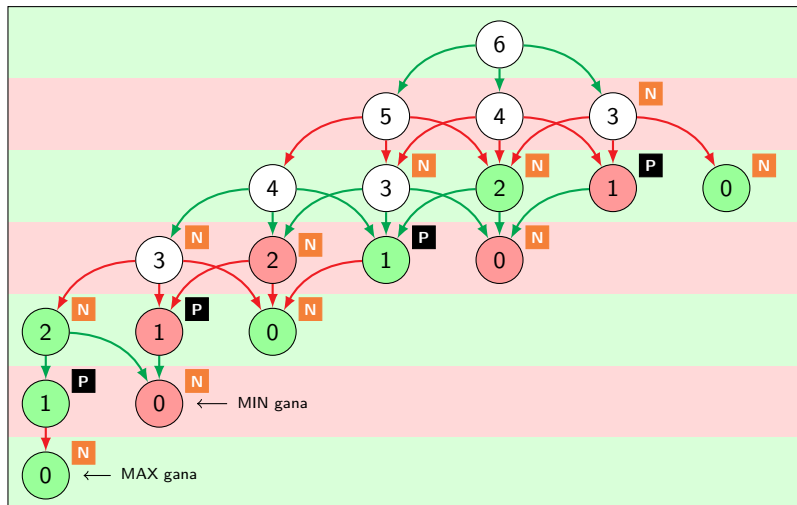
## El juego de los corchos a partir del estado 6



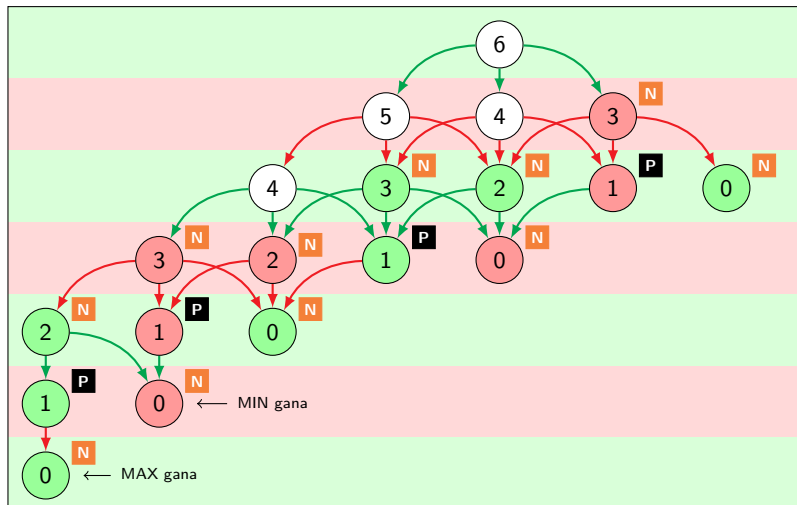
## El juego de los corchos a partir del estado 6



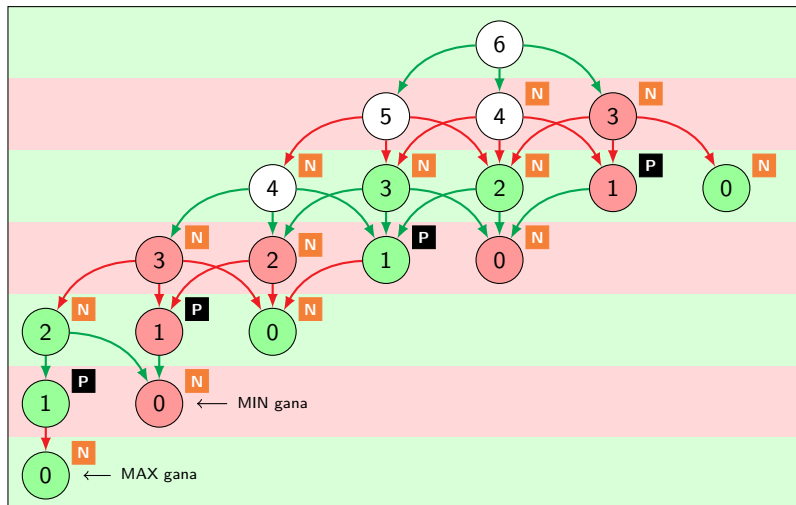
## El juego de los corchos a partir del estado 6



## El juego de los corchos a partir del estado 6

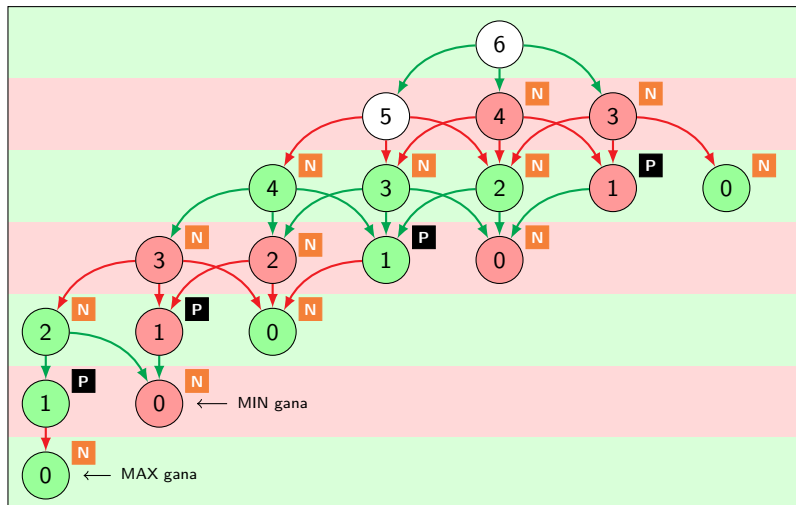


## El juego de los corchos a partir del estado 6

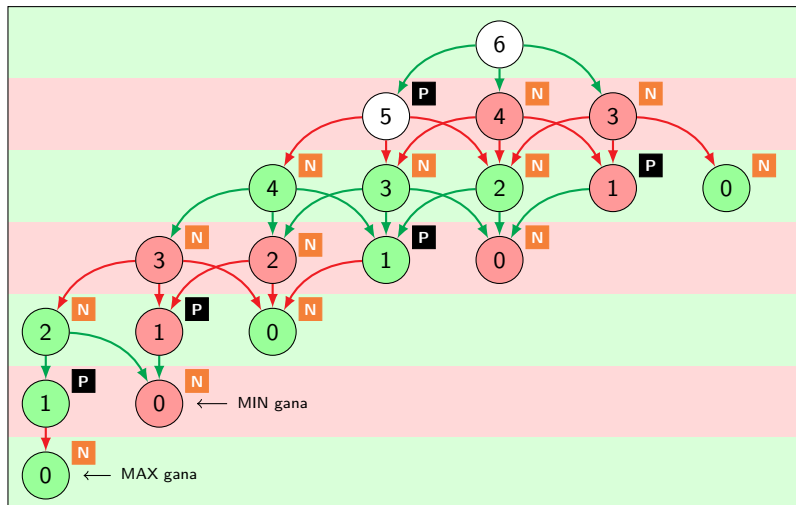




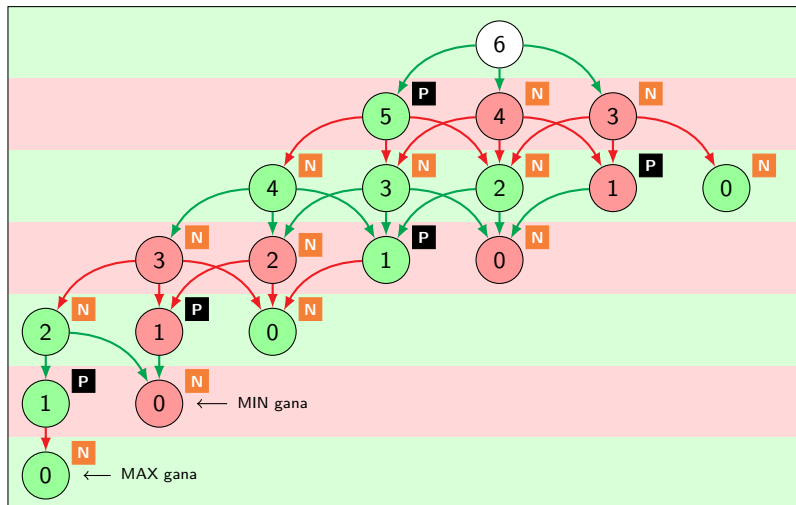
## El juego de los corchos a partir del estado 6



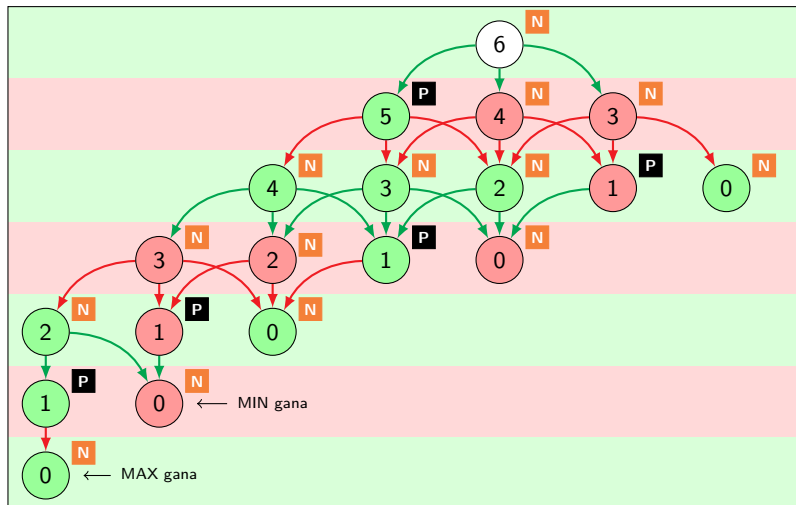
## El juego de los corchos a partir del estado 6



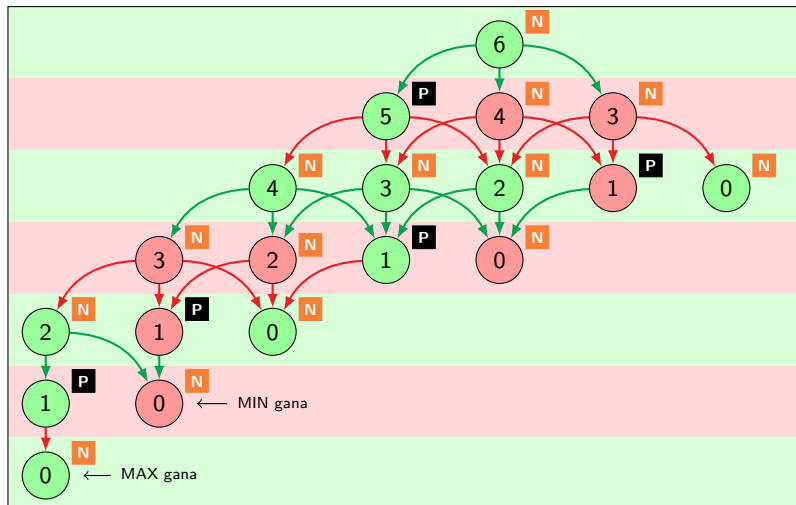
## El juego de los corchos a partir del estado 6



## El juego de los corchos a partir del estado 6

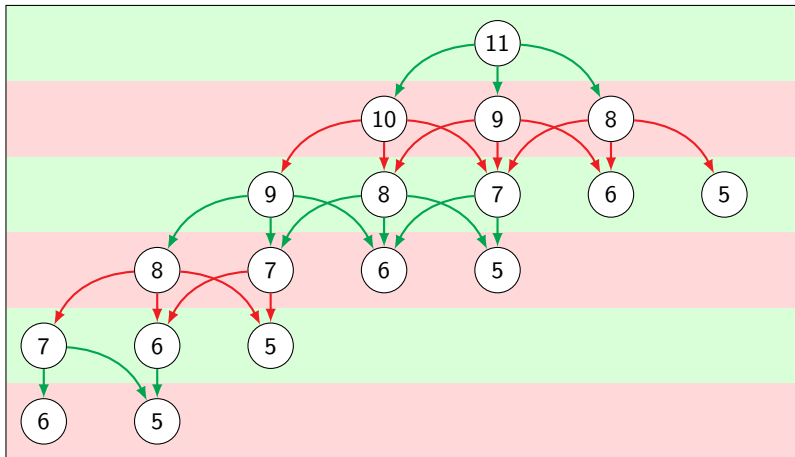


## El juego de los corchos a partir del estado 6

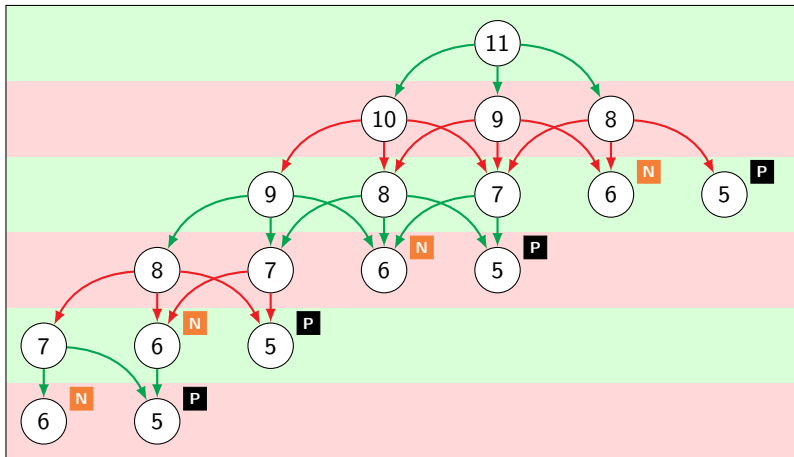


# El juego de los corchos a partir del estado 11

## El juego de los corchos a partir del estado 11

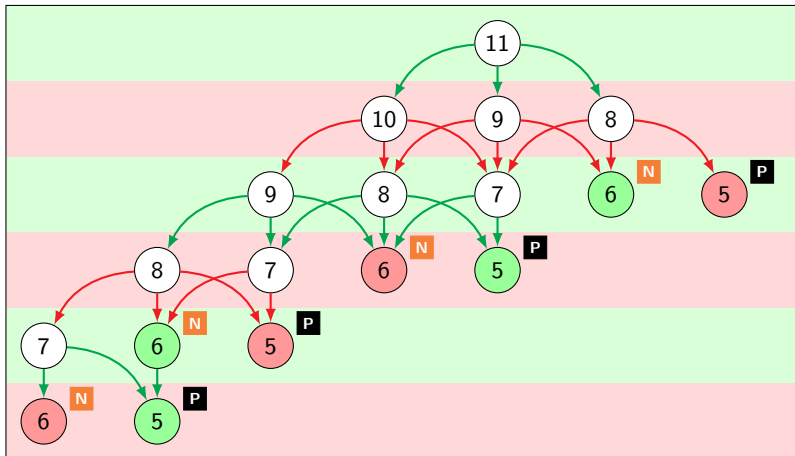


## El juego de los corchos a partir del estado 11

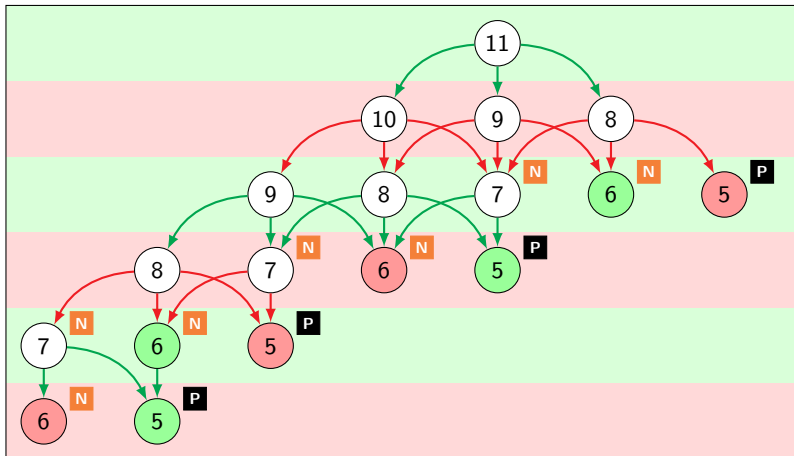




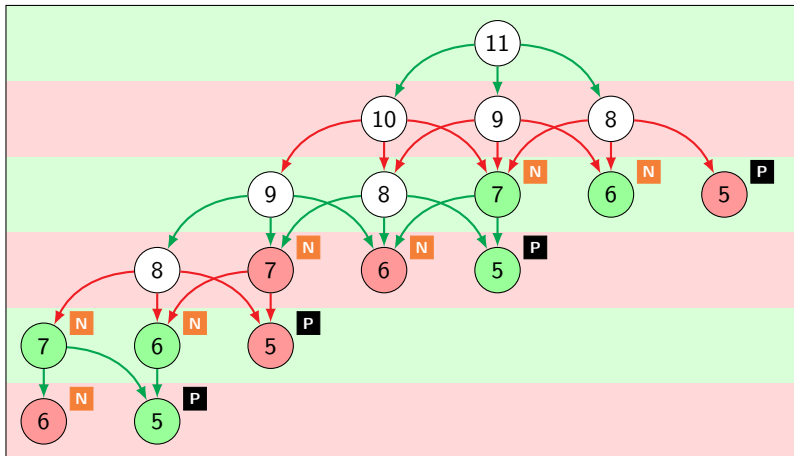
## El juego de los corchos a partir del estado 11



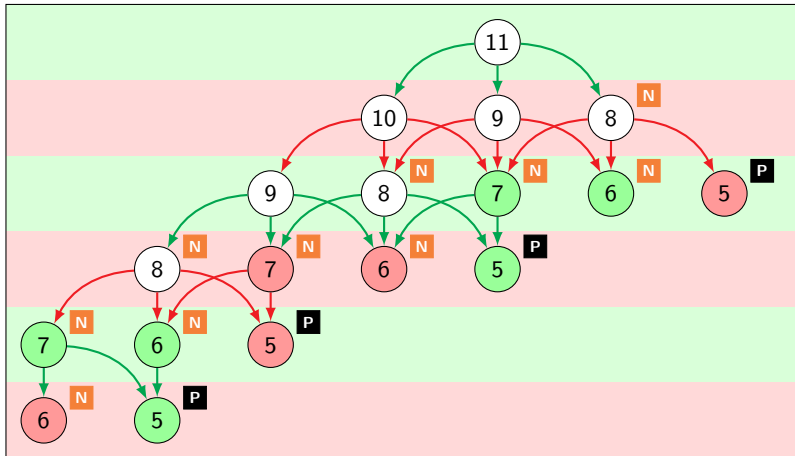
## El juego de los corchos a partir del estado 11



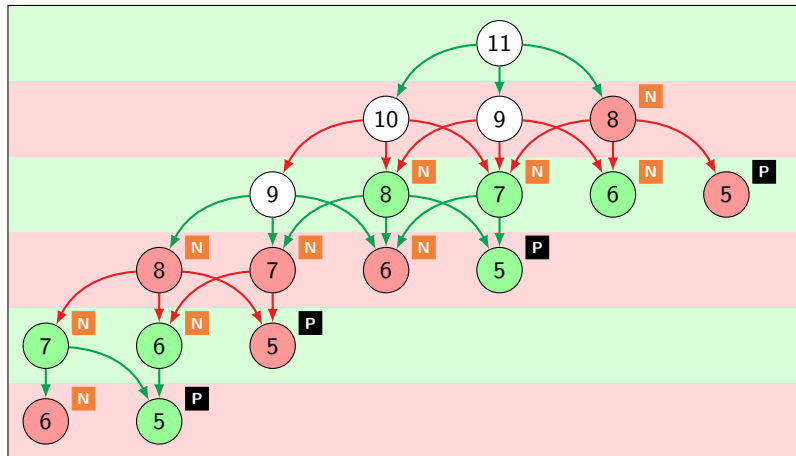
## El juego de los corchos a partir del estado 11



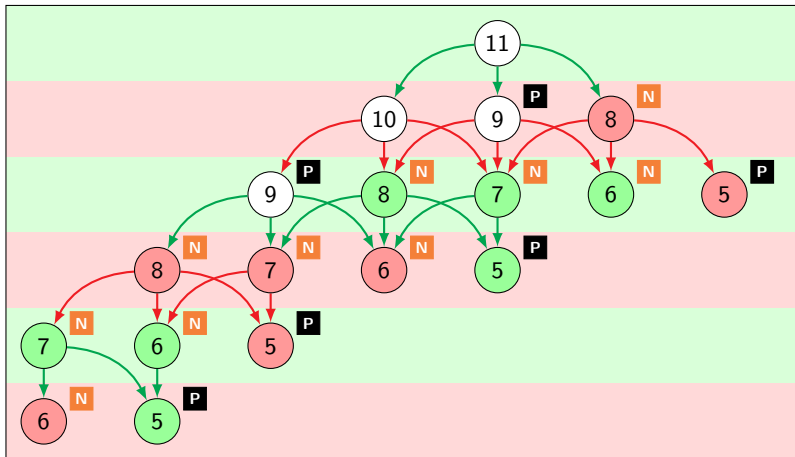
## El juego de los corchos a partir del estado 11



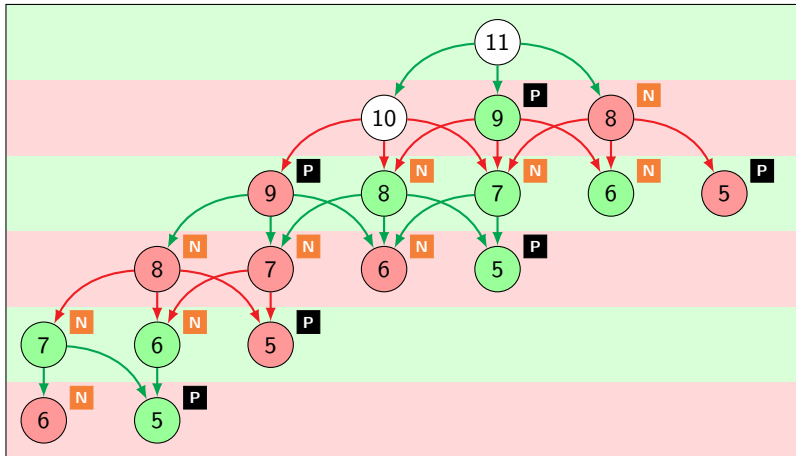
## El juego de los corchos a partir del estado 11



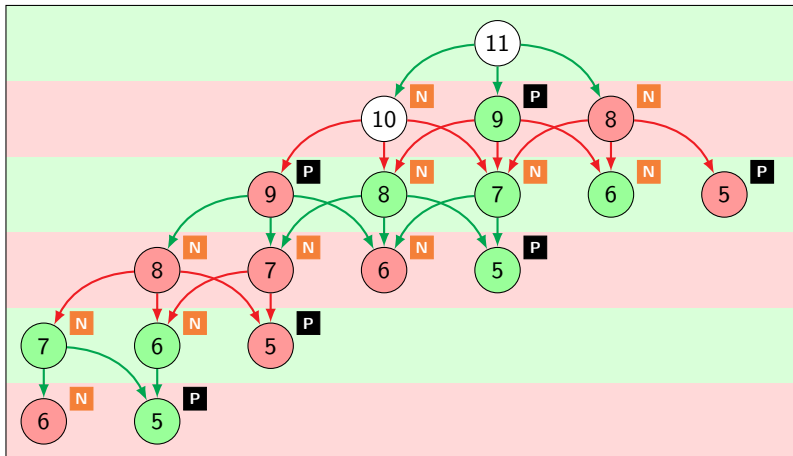
## El juego de los corchos a partir del estado 11



## El juego de los corchos a partir del estado 11

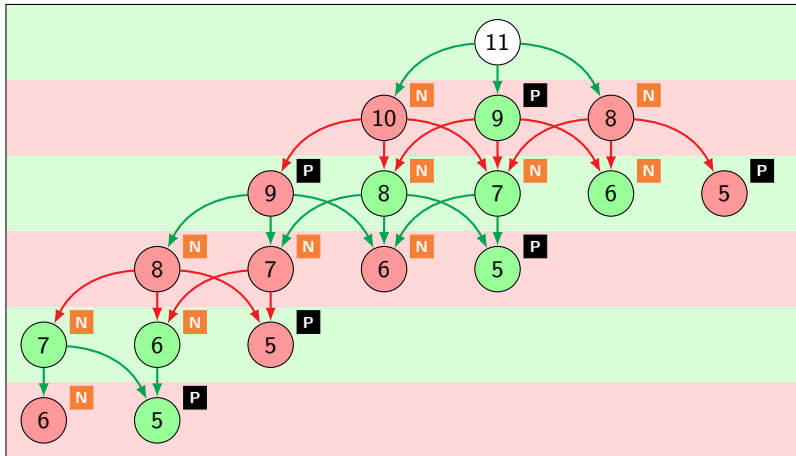


## El juego de los corchos a partir del estado 11

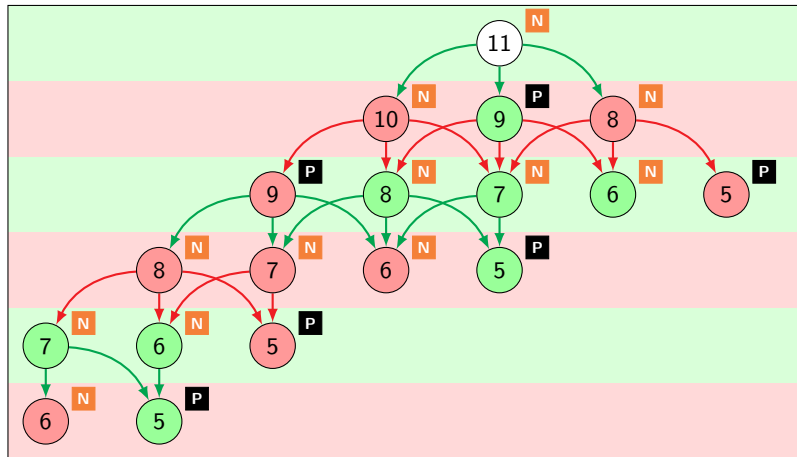




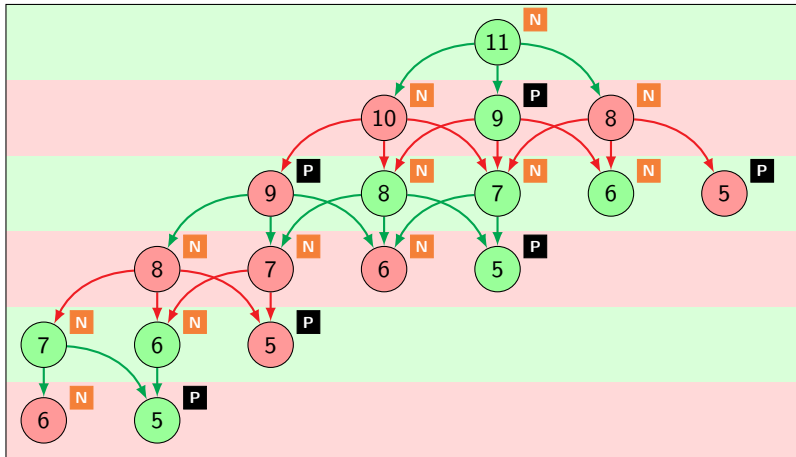
## El juego de los corchos a partir del estado 11



## El juego de los corchos a partir del estado 11



## El juego de los corchos a partir del estado 11



# Un algoritmo para el juego de los corchos

```
1.  algoritmo Escabio (int e, n)    // e como antes; n es 1 (MAX) o -1 (MIN)
2.  if (e == 0) {
3.      return n                    // si no hay más corchos, el jugador de turno gana
4.  } else {
5.      sig = 1                      // siguiente jugada
6.      val = -1 * n
7.      posa = false
8.      while (sig ≤ 3) && (sig ≤ e) && !posa) {
9.          e = e - sig
10.         if (n == 1) {
11.             val = max(val, Escabio(e, -1 * n))
12.         } else {
13.             val = min(val, Escabio(e, -1 * n))
14.         }
15.         if (n * val) = 1          // n = 1 y val = 1 o n = -1 and val = -1
16.             posa = true
17.         } else {
18.             e = e + sig
19.         }
20.         sig = sig + 1
21.     }
22. }
23. return val
```

## 1 Repaso de la clase anterior

## 2 Juegos

- El algoritmo Min-Max
- El juego de los corchos
- Back to Classics. El juego del ta-te-ti
- La poda alfa-beta

## 3 Ejercicios propuestos

# *Back to Classics.* El juego del ta-te-ti

## *Back to Classics.* El juego del ta-te-ti

- El ta-te-ti es un juego que se juega con papel y lápiz para dos jugadores que marcan por turno los espacios en blanco de una grilla de tres por tres con O o X.

## *Back to Classics.* El juego del ta-te-ti

- El ta-te-ti es un juego que se juega con papel y lápiz para dos jugadores que marcan por turno los espacios en blanco de una grilla de tres por tres con O o X.
- Una vez que se marca un espacio, el espacio queda lleno y no puede ser marcado de nuevo. Tampoco se puede borrar el símbolo marcado.



## *Back to Classics.* El juego del ta-te-ti

- El ta-te-ti es un juego que se juega con papel y lápiz para dos jugadores que marcan por turno los espacios en blanco de una grilla de tres por tres con O o X.
- Una vez que se marca un espacio, el espacio queda lleno y no puede ser marcado de nuevo. Tampoco se puede borrar el símbolo marcado.
- En nuestro ejemplo, *Max* marcará con O y *Min* marcará con X.

## *Back to Classics.* El juego del ta-te-ti

- El ta-te-ti es un juego que se juega con papel y lápiz para dos jugadores que marcan por turno los espacios en blanco de una grilla de tres por tres con O o X.
- Una vez que se marca un espacio, el espacio queda lleno y no puede ser marcado de nuevo. Tampoco se puede borrar el símbolo marcado.
- En nuestro ejemplo, *Max* marcará con O y *Min* marcará con X.
- El jugador que pueda alinear tres símbolos en alguna fila, columna o diagonal, gana el juego.

## *Back to Classics.* El juego del ta-te-ti

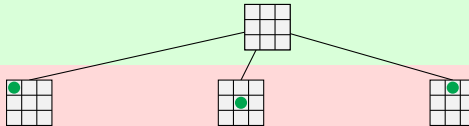
- El ta-te-ti es un juego que se juega con papel y lápiz para dos jugadores que marcan por turno los espacios en blanco de una grilla de tres por tres con O o X.
- Una vez que se marca un espacio, el espacio queda lleno y no puede ser marcado de nuevo. Tampoco se puede borrar el símbolo marcado.
- En nuestro ejemplo, *Max* marcará con O y *Min* marcará con X.
- El jugador que pueda alinear tres símbolos en alguna fila, columna o diagonal, gana el juego.
- El juego termina con un empate si ambos bandos juegan correctamente.



# El juego del Ta-te-ti



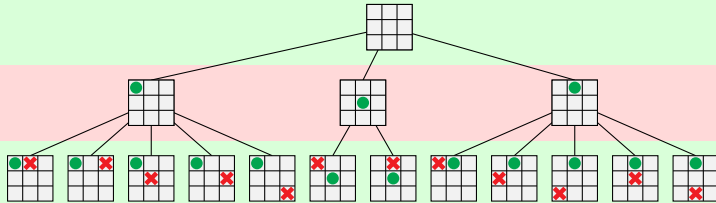
- ☐ MAX gana
- ☐ Juego empatado



# El juego del Ta-te-ti



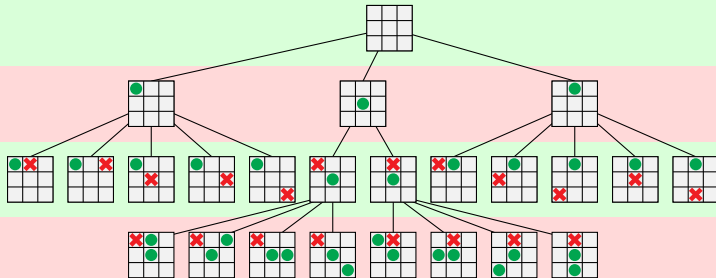
-  MAX gana
-  Juego empatado

# El juego del Ta-te-ti



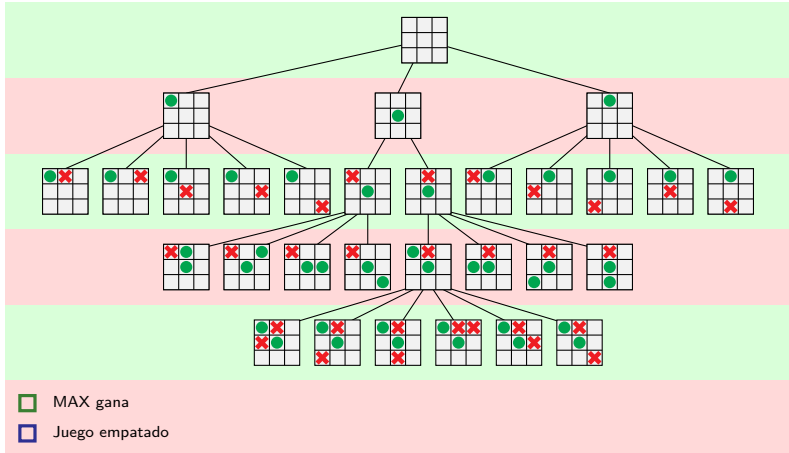
-  MAX gana
-  Juego empatado

# El juego del Ta-te-ti



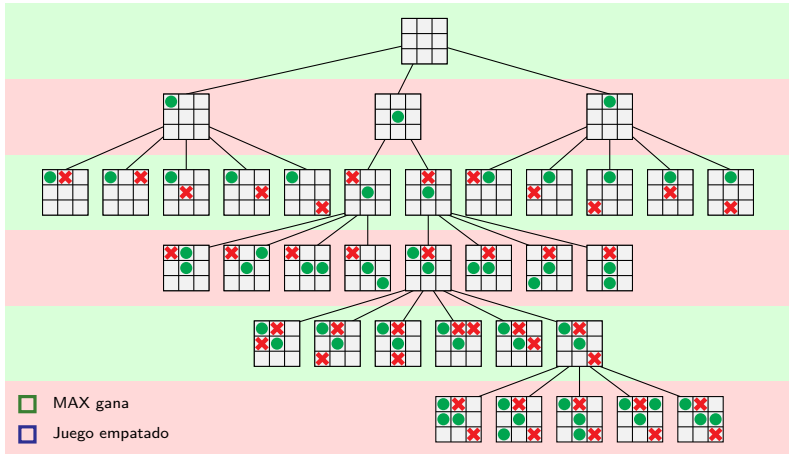
- MAX gana
- Juego empatado

# El juego del Ta-te-ti

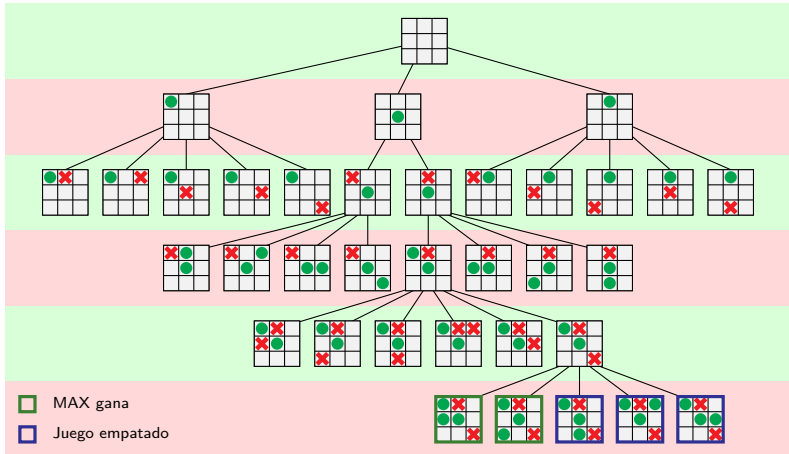




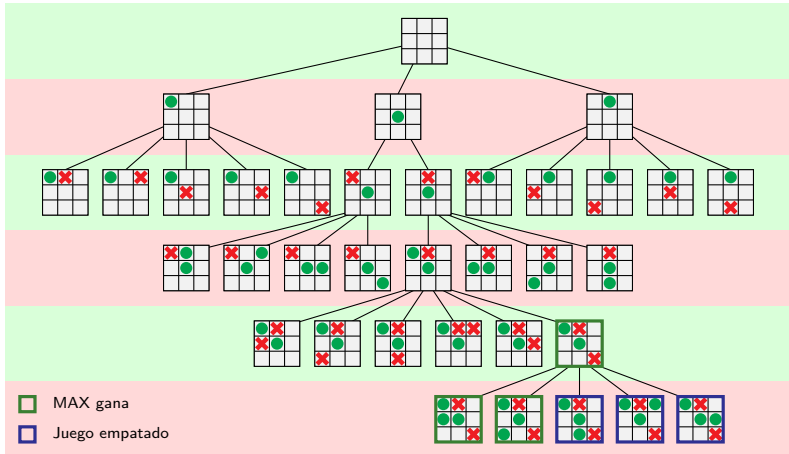
# El juego del Ta-te-ti



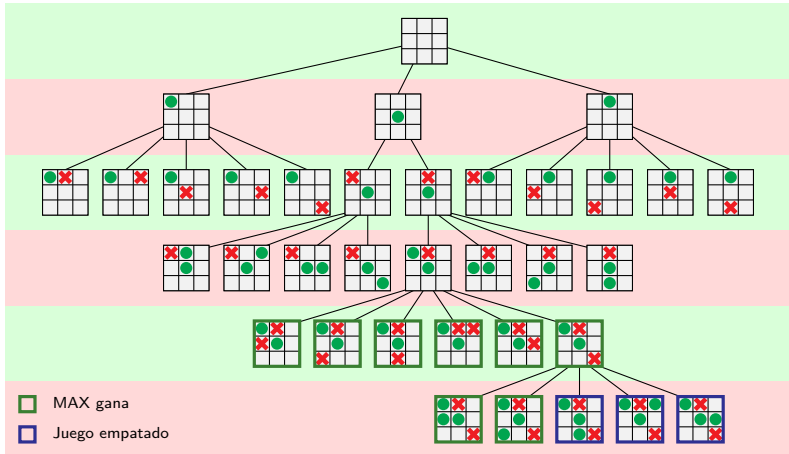
# El juego del Ta-te-ti



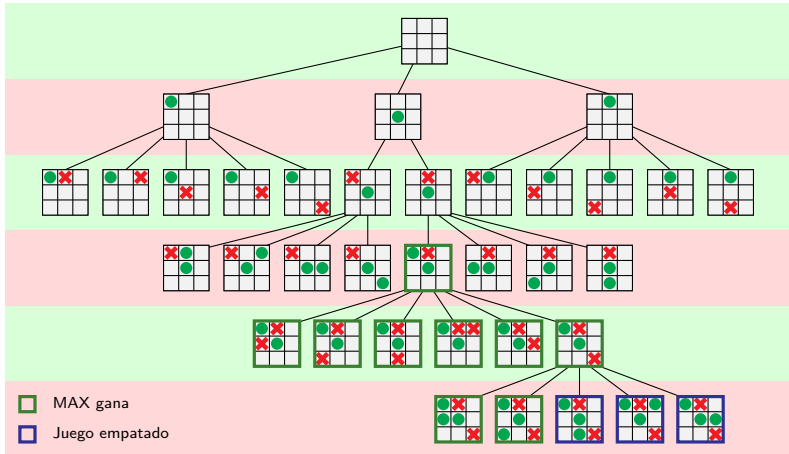
# El juego del Ta-te-ti



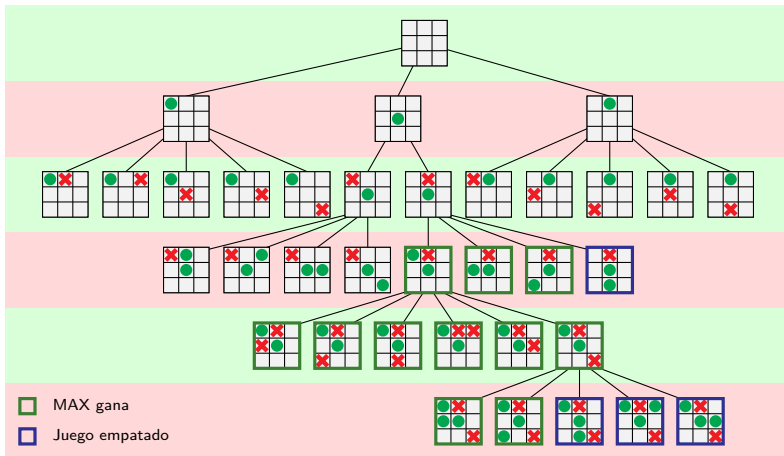
# El juego del Ta-te-ti



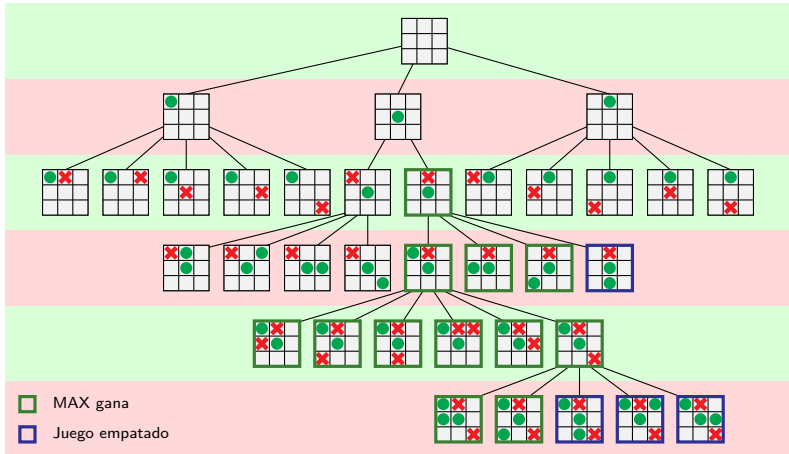
# El juego del Ta-te-ti



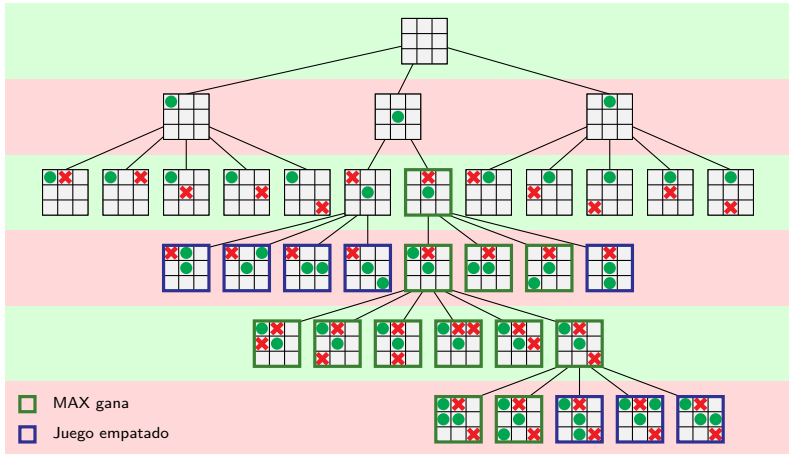
# El juego del Ta-te-ti



# El juego del Ta-te-ti

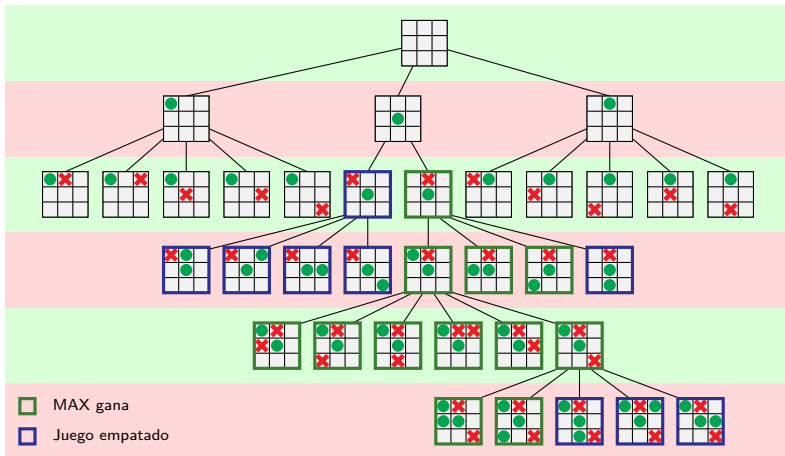


# El juego del Ta-te-ti

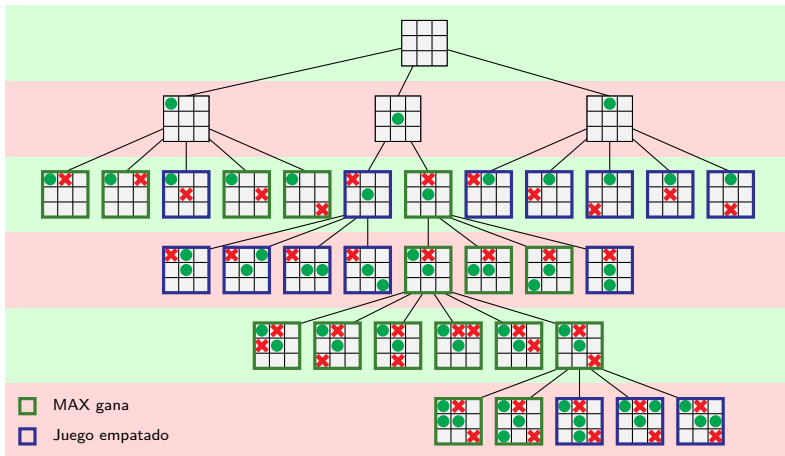




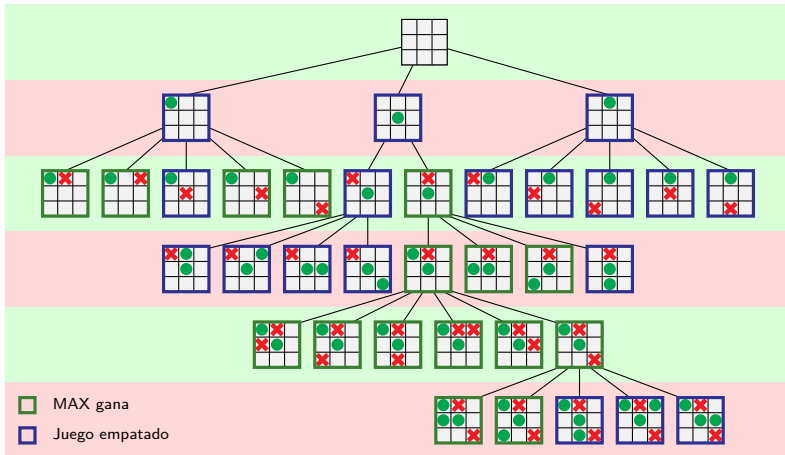
# El juego del Ta-te-ti



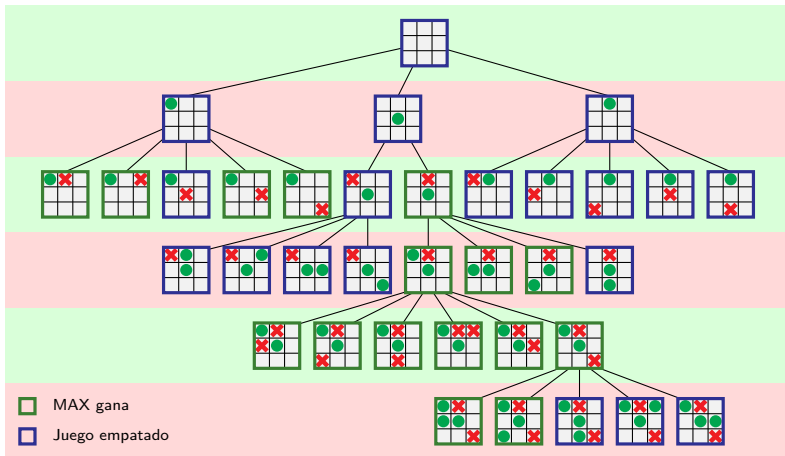
# El juego del Ta-te-ti



# El juego del Ta-te-ti



# El juego del Ta-te-ti



## 1 Repaso de la clase anterior

## 2 Juegos

- El algoritmo Min-Max
- El juego de los corchos
- Back to Classics. El juego del ta-te-ti
- La poda alfa-beta

## 3 Ejercicios propuestos

# La poda alfa-beta

# La poda alfa-beta

- No es un nuevo algoritmo, sino una optimización del algoritmo *MinMax*.

# La poda alfa-beta

- No es un nuevo algoritmo, sino una optimización del algoritmo *MinMax*.
- Cada nodo está equipado con dos nuevos parámetros,  $\alpha$  y  $\beta$ .



# La poda alfa-beta

- No es un nuevo algoritmo, sino una optimización del algoritmo *MinMax*.
- Cada nodo está equipado con dos nuevos parámetros,  $\alpha$  y  $\beta$ .
- $\alpha$  es el mejor resultado que puede obtener *Max* en esa rama y  $\beta$  es el mejor resultado que *Min* puede obtener.

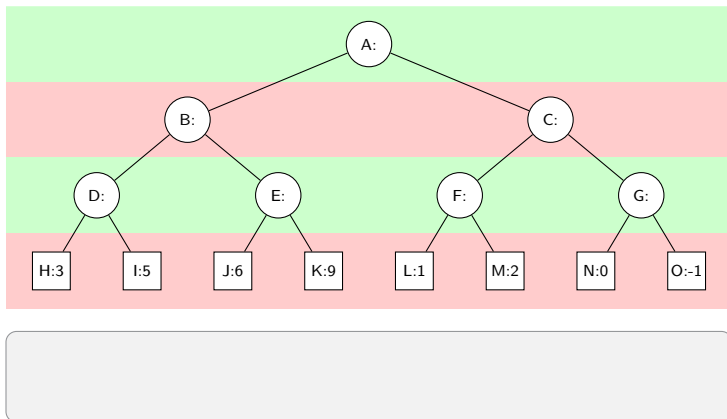
# La poda alfa-beta

- No es un nuevo algoritmo, sino una optimización del algoritmo *MinMax*.
- Cada nodo está equipado con dos nuevos parámetros,  $\alpha$  y  $\beta$ .
- $\alpha$  es el mejor resultado que puede obtener *Max* en esa rama y  $\beta$  es el mejor resultado que *Min* puede obtener.
- Los parámetros  $\alpha$  y  $\beta$  se pasan hacia abajo, pero no hacia arriba.

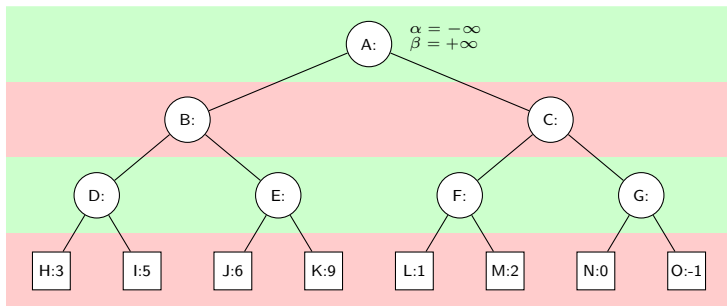
## La poda alfa-beta

- No es un nuevo algoritmo, sino una optimización del algoritmo *MinMax*.
- Cada nodo está equipado con dos nuevos parámetros,  $\alpha$  y  $\beta$ .
- $\alpha$  es el mejor resultado que puede obtener *Max* en esa rama y  $\beta$  es el mejor resultado que *Min* puede obtener.
- Los parámetros  $\alpha$  y  $\beta$  se pasan hacia abajo, pero no hacia arriba.
- La idea principal es que si  $\alpha \geq \beta$ , entonces los dos jugadores saben que su oponente no irá por esa rama del árbol, pues tiene una opción mejor más arriba, en el camino hacia la raíz.

## La poda alfa-beta. Un Ejemplo

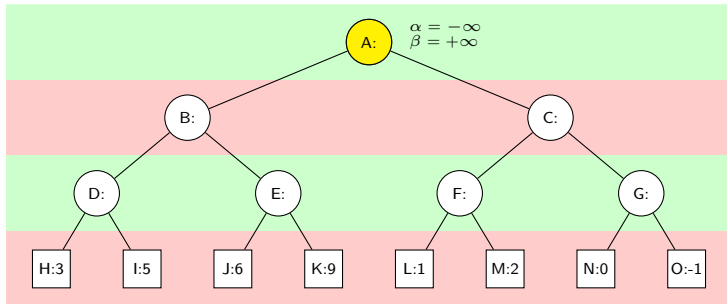


# La poda alfa-beta. Un Ejemplo



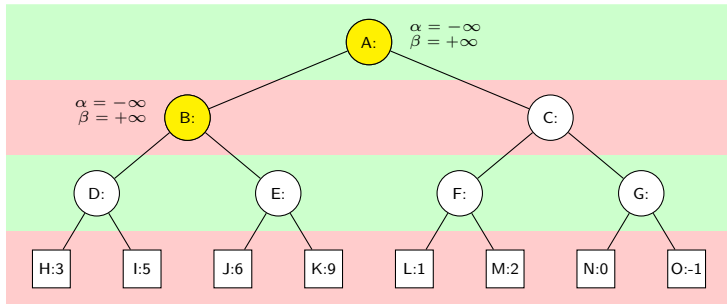
El proceso comienza en **A**  
Valores iniciales:  $\alpha = -\infty$  y  $\beta = \infty$ .

## La poda alfa-beta. Un Ejemplo



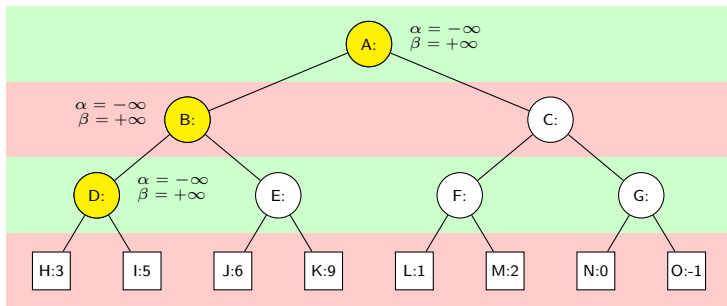
*Max* elige el máximo entre *B* y *C*; se evalúa primero *B*.

## La poda alfa-beta. Un Ejemplo



Los valores de  $\alpha$  y  $\beta$  se pasan hacia abajo a  $B$ .  
 $\text{Min}$  elige el mínimo entre  $D$  and  $E$ ; se evalúa primero  $D$ .

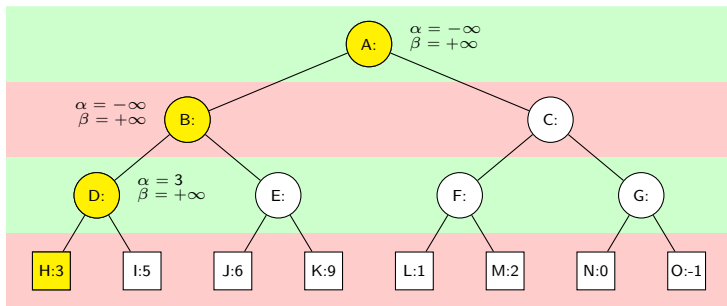
# La poda alfa-beta. Un Ejemplo



Los valores de  $\alpha$  y  $\beta$  se pasan hacia abajo a  $D$ .  
 $\text{Max}$  elige el máximo entre  $H$  and  $I$ ; se evalúa primero  $H$ .



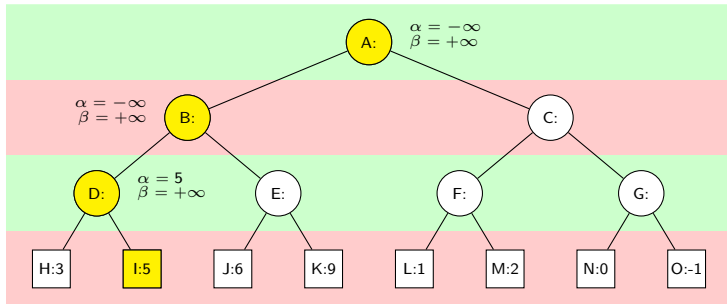
# La poda alfa-beta. Un Ejemplo



El nodo *H* devuelve 3.

El valor de  $\alpha$  en *D* se actualiza a 3.

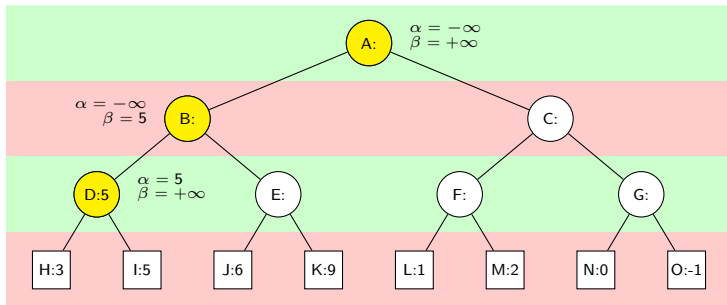
## La poda alfa-beta. Un Ejemplo



El nodo *I* devuelve 5.

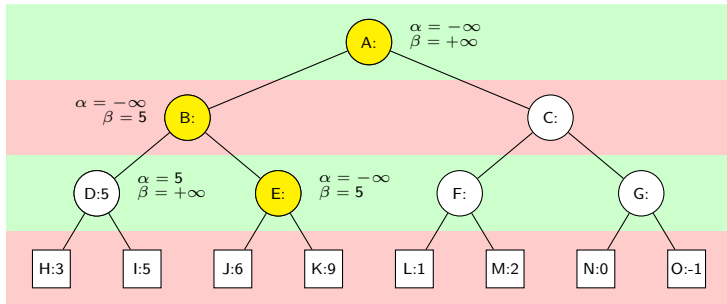
El valor de  $\alpha$  en *D* se actualiza a 5.

## La poda alfa-beta. Un Ejemplo



El nodo *D* devuelve  $\max(3, 5) = 5$ .  
El valor de  $\beta$  en *B* se actualiza a 5.

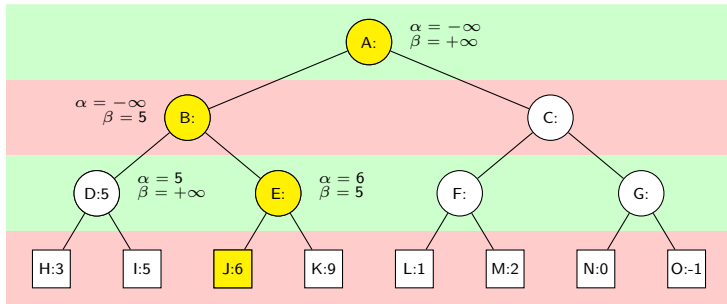
# La poda alfa-beta. Un Ejemplo



Los valores de  $\alpha$  y  $\beta$  se pasan hacia abajo a  $E$ .

$\text{Max}$  elige el máximo entre  $J$  and  $K$ ; se evalúa primero  $J$ .

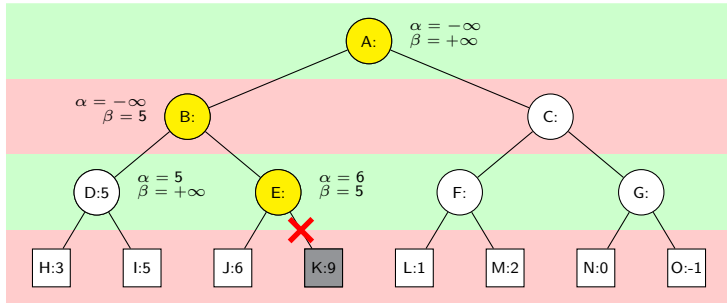
## La poda alfa-beta. Un Ejemplo



El nodo *J* devuelve 6.

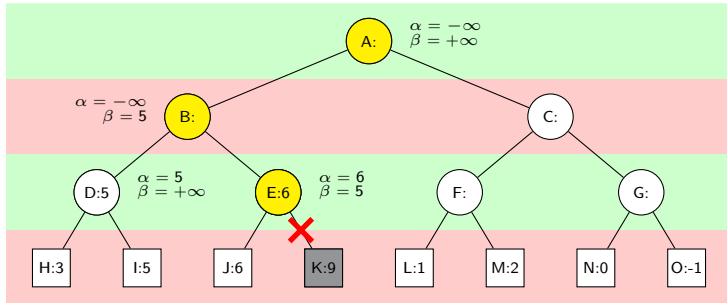
El valor de  $\alpha$  en *E* se actualiza a 6.

# La poda alfa-beta. Un Ejemplo



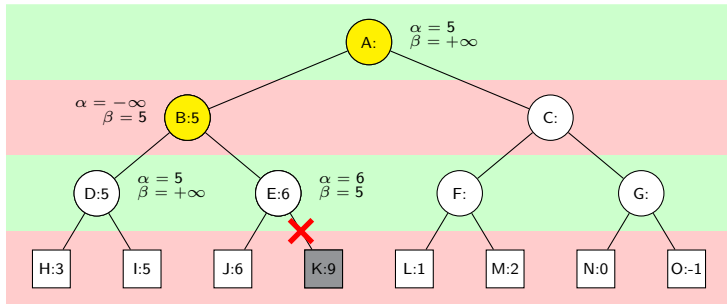
Tenemos  $\alpha \geq \beta$   
La rama se poda.

# La poda alfa-beta. Un Ejemplo



El nodo *E* devuelve 6.

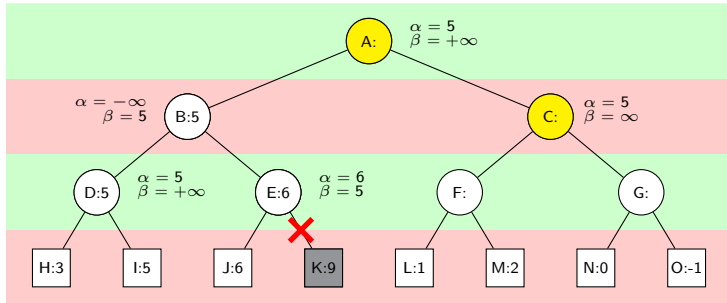
## La poda alfa-beta. Un Ejemplo



El nodo *B* devuelve  $\min(5, 6) = 5$ .  
El valor de  $\alpha$  en *A* se actualiza a 5.

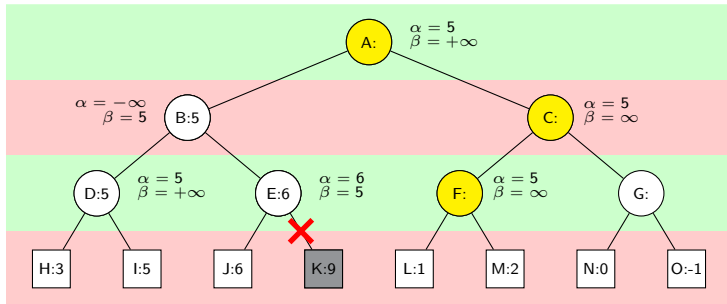


# La poda alfa-beta. Un Ejemplo



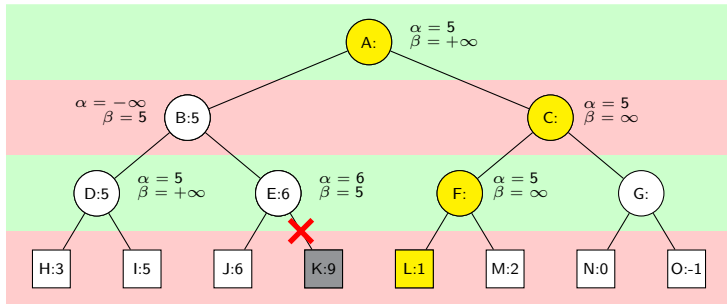
Los valores de  $\alpha$  y  $\beta$  se pasan hacia abajo a C.  
*Min* elige el mínimo entre F and G; se evalúa primero F.

## La poda alfa-beta. Un Ejemplo



Los valores de  $\alpha$  y  $\beta$  se pasan hacia abajo a  $F$ .  
 $\text{Max}$  elige el máximo entre  $L$  and  $M$ ; se evalúa primero  $L$ .

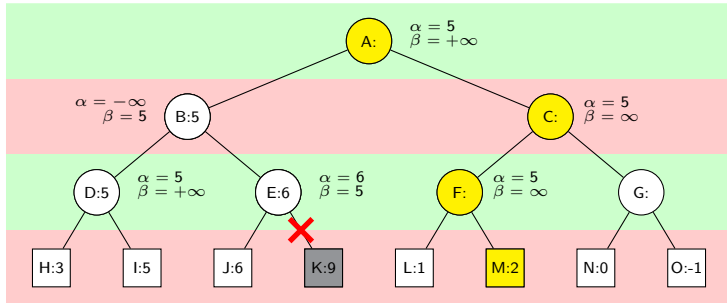
## La poda alfa-beta. Un Ejemplo



El nodo *L* devuelve 1.

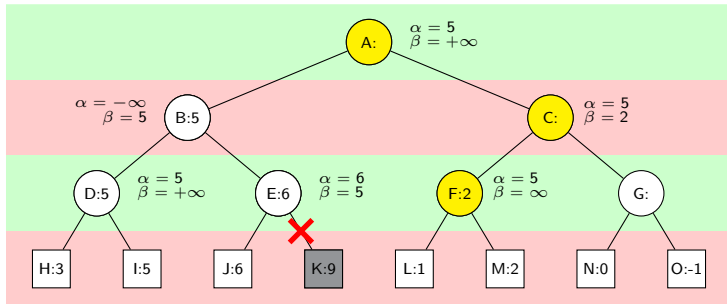
El valor de  $\alpha$  no cambia en *F*.

# La poda alfa-beta. Un Ejemplo



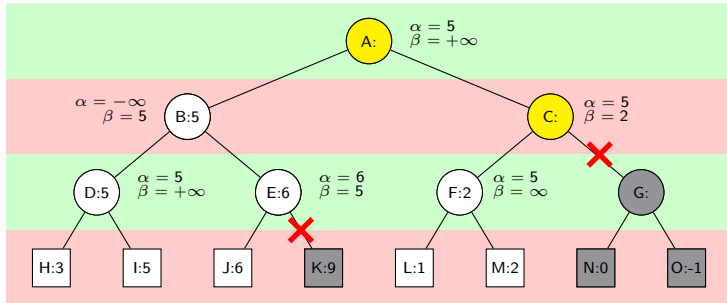
El nodo *M* devuelve 2.  
El valor de  $\alpha$  no cambia en *F*.

## La poda alfa-beta. Un Ejemplo



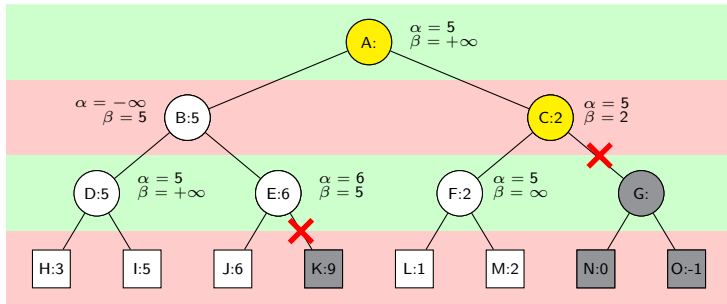
El nodo *F* devuelve  $\min(1, 2) = 2$ .  
El valor de  $\beta$  en *C* se actualiza a 2.

# La poda alfa-beta. Un Ejemplo



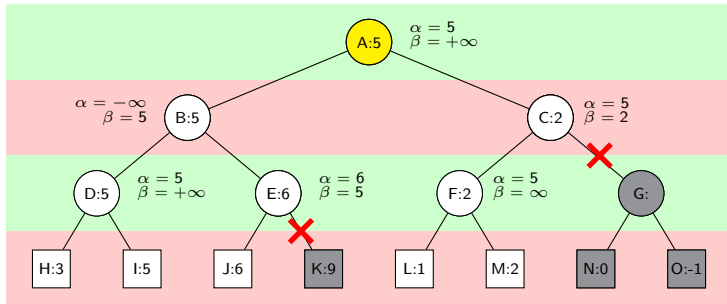
Tenemos  $\alpha \geq \beta$   
La rama se poda.

# La poda alfa-beta. Un Ejemplo



El nodo C devuelve 2.

## La poda alfa-beta. Un Ejemplo



El nodo A devuelve  $\max(5, 2) = 5$ .

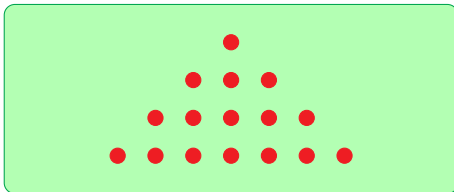


## El algoritmo alfa-beta

```
1  algoritmo AB
2  input e: estado, n: nivel,  $\alpha, \beta$ :integer // estado y nivel; inicialmente,  $\alpha = -\infty$  y  $\beta = +\infty$ 
3  if hoja(e, n)
4      return result(e, n) // devuelve el valor
5  else
6      if n = MAX
7          val  $\leftarrow -\infty$ 
8      else
9          val  $\leftarrow +\infty$ 
10     endif
11     foreach sig  $\leftarrow$  son(e) do
12         if n = MAX
13             val  $\leftarrow$  max(val, AB(sig, MIN,  $\alpha, \beta$ ))
14              $\alpha \leftarrow$  max(val,  $\alpha$ )
15         else
16             val  $\leftarrow$  min(val, AB(sig, MAX,  $\alpha, \beta$ ))
17              $\beta \leftarrow$  min(val,  $\beta$ )
18         endif
19         if  $\beta \leq \alpha$ 
20             break
21         endif
22     endforeach
23 endif
24 return val
```

# Ejercicios propuestos 1

- 1 En el juego de *Nim* dos participantes se turnan para sacar objetos de varios grupos o pilas. En cada movida, un participante debe sacar por lo menos un objeto, aunque puede retirar tantos como quiera siempre que sean todos del mismo grupo.



Un ejemplo con tres grupos de objetos.

Escriba un programa que determine qué jugador gana el juego de Nim game de la figura. Los grupos están dispuestos horizontalmente.

- 2 Suponga que lo desafían a un partido del juego de los corchos con un número arbitrario, pero conocido de antemano, decorchos y que le ofrecen la opción de comenzar el juego o hacer la segunda movida. Diseñe una estrategia ganadora *greedy*.
- 3 ¿Qué cambiaría en el juego de los 11 corchos si el ganador fuera quien retira el último corcho?
- 4 Dibuje el árbol del juego de los 11 corchos según las reglas del ejercicio anterior.

## Ejercicios propuestos 2

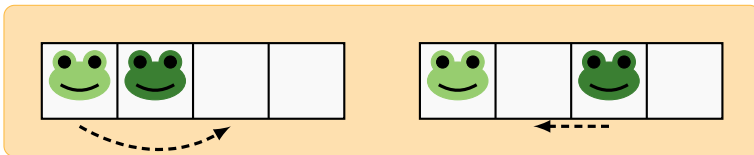
- 5 Un juego muy simple de dos jugadores. Se comienza con un grupo de  $n$  objetos (monedas, fósforos, lo que sea.) En cada turno, un jugador debe dividir el grupo existente en dos. Por supuesto, un grupo de un solo elemento no puede ser dividido. El jugador que no puede dividir ningún grupo pierde el juego. Dibuje el árbol del juego comenzando por un grupo de 6 objetos. Determine cuáles son los nodos  $N$  y los nodos  $P$ . ¿Qué conclusiones puede extraer?
- 6 Considere ahora la siguiente variante del juego del ejercicio anterior: cada jugador debe dividir cada grupo en dos grupos *desiguales*. El juego termina con la derrota del jugador que ya no puede dividir ningún grupo, porque sólo quedan grupos de 1 o 2 objetos. Dibuje el árbol de este juego para 8, 9 y 10 objects.
- 7 Y todavía otro juego simple para dos jugadores. Cada jugador elige una jugada en su turno. El "jugador de filas" elige una fila entre  $T$ ,  $M$  y  $B$  y el "jugador de columnas" elige una columna entre  $L$ ,  $C$  y  $R$ . El resultado del juego es la combinación de ambas jugadas según la siguiente tabla de resultados, donde el primer número corresponde a la ganancia del "jugador de filas" y el segundo a la ganancia del "jugador de columnas."

	L	C	R
T	3, 1	5, -5	2, -20
M	5, 0	-10, 1	3, -2
B	3, 5	4, 4	-22, 2

El objetivo de ambos jugadores es maximizar su ganancia independientemente del resultado obtenido por el rival. Construya un árbol para este juego utilizando una estrategia similar a *MinMax*.

## Ejercicios propuestos 3

- 1 El juego de las ranas y los sapos. Alicia ha entrenado un grupo de ranas y Bruno un grupo de sapos para jugar al siguiente juego. En cada turno, uno de los jugadores persuade a alguna de sus criaturas para que se muevan una posición o salten por encima de un rival a la siguiente posición libre. Las ranas se mueven siempre hacia la derecha y los sapos hacia la izquierda.

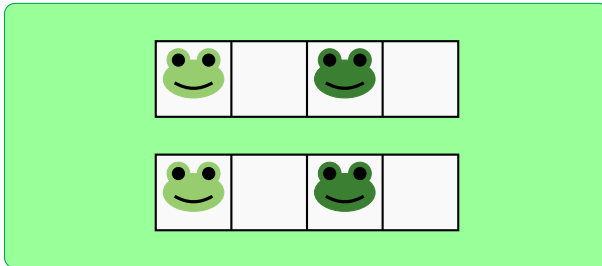


Dos ejemplos de los dos tipos de movidas legales.

El jugador a quien le toca jugar y no tiene ningún batracio móvil (por ejemplo, porque todos llegaron al otro extremo), pierde el juego. Se pide:

- Dibuje el árbol del juego para el estado inicial que se muestra en la página siguiente. No dibuje todo el árbol; utilice algún método de poda (por ejemplo,
- Estime la reducción de la cantidad de nodos generados cuando se poda y cuando se genera el árbol completo.

## Ejercicios propuestos 4



La posición inicial para el juego de las ranas y los sapos.