

Competición

Sistemas Inteligentes Distribuidos

Sergio Alvarez

Javier Vázquez

Bibliografía

- *Artificial intelligence: a modern approach* (Russell & Norvig), cap. 6, 17
- *Multiagent systems: algorithmic, game-theoretic, and logical foundations* (Shoham & Leyton-Brown), cap. 5
- *Multi-Agent Reinforcement Learning* (Albrecht et al.), cap. 9
- *Reinforcement Learning: An Introduction* (Sutton & Barto), cap. 2, 5
- *Game Theory, Alive* (Karlin & Peres), cap. 6, 12

Introducción

Cooperación

¿Qué es competición?

- **La competencia es un tipo de coordinación entre agentes antagonistas que compiten entre sí o que son egoístas**
- El grado de éxito en la negociación (para un agente dado) se puede medir por
 - La capacidad del agente para maximizar su propio beneficio
 - La capacidad de no tener en cuenta el beneficio de los otros agentes
 - La capacidad de minimizar el beneficio de otros agentes

Posibles enfoques

- **Económico:** viendo los sistemas multiagente como economías con objetos y precios de oferta y demanda
- **Parte del entorno:** los adversarios son simplemente parte del entorno y contribuyen a su parte dinámica
- **Problema de búsqueda:** la mejor estrategia frente a las estrategias del adversario se puede extraer a partir de una búsqueda en un espacio de estados

Teorías y modelos

- **Juegos en forma extensiva**
 - **Minimax**
 - **Expectiminimax**
 - **Monte Carlo Tree Search**
- **Negociación**
 - **Bargaining**
 - **Argumentación**
- **Diseño de mecanismos**

Juegos en forma extensiva

Competición

De forma normal a forma extensiva

- La representación en forma normal permite razonar sobre las estrategias en base a supuestos de simultaneidad de acciones
 - Una variante son los juegos en forma repetida, que consisten en múltiples ejecuciones de un mismo juego en forma normal
- Los **juegos en forma extensiva** permiten formalizar situaciones en las que los agentes participan en secuencia, por turnos
 - Existen métodos para reducir juegos en forma extensiva a forma normal, por lo que es posible encontrar e.g. equilibrios de Nash
- La representación en forma extensiva es, básicamente, una representación **en forma de árbol**
 - Estos árboles pueden ser infinitos, pero nos centraremos en la versión finita

Juegos en forma extensiva (información perfecta)

- Un **juego en forma extensiva con información perfecta** es un árbol que se define como una tupla $G = \langle N, A, H, Z, \mathcal{X}, \rho, u \rangle$:
 - N es un conjunto de agentes
 - A es un conjunto global de acciones
 - H es un conjunto de nodos estado no terminales
 - Z es un conjunto de nodos estado terminales, $H \cap Z = \emptyset$
 - $\mathcal{X}: H \rightarrow 2^A$ es la función de acción, que asigna un conjunto de acciones posibles a cada nodo estado
 - $\rho: H \rightarrow N$ es la función de agente, que asigna un jugador $i \in N$ a cada nodo no terminal, representando al agente que puede elegir acción
 - $u = (u_1, \dots, u_n)$, donde $u_i: Z \rightarrow \mathbb{R}$ es una función que asigna una utilidad a cada agente i en cada nodo terminal Z

Juegos en forma extensiva (información perfecta)

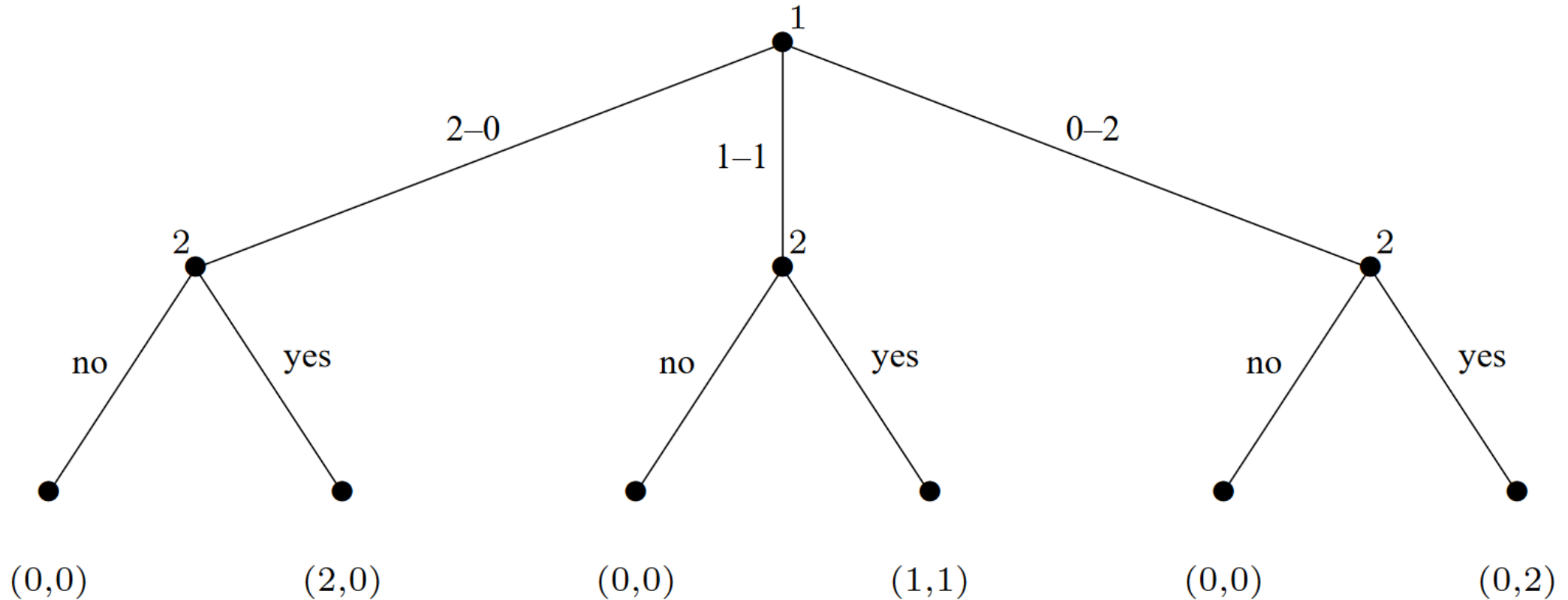
- En general, diremos que un juego es de **información perfecta** cuando el entorno es **totalmente observable**
- Los nodos en H y Z se relacionan en forma de árbol
- Es posible identificar la **historia del juego como la secuencia de acciones** que se han escogido desde el nodo raíz hasta el nodo actual

Ejemplo: *sharing game*

- Dos hermanos (agentes 1 y 2) reciben dos regalos idénticos e indivisibles
- Siguen el siguiente protocolo para repartírselos:
 - 1 elige entre: 1 se queda los dos (**2-0**), 2 se queda los dos (**0-2**), se reparten uno cada uno (**1-1**)
 - 2 elige entre: aceptar o rechazar el reparto
 - Si acepta, se ejecuta el reparto
 - Si no acepta, nadie recibe nada (**0-0**)

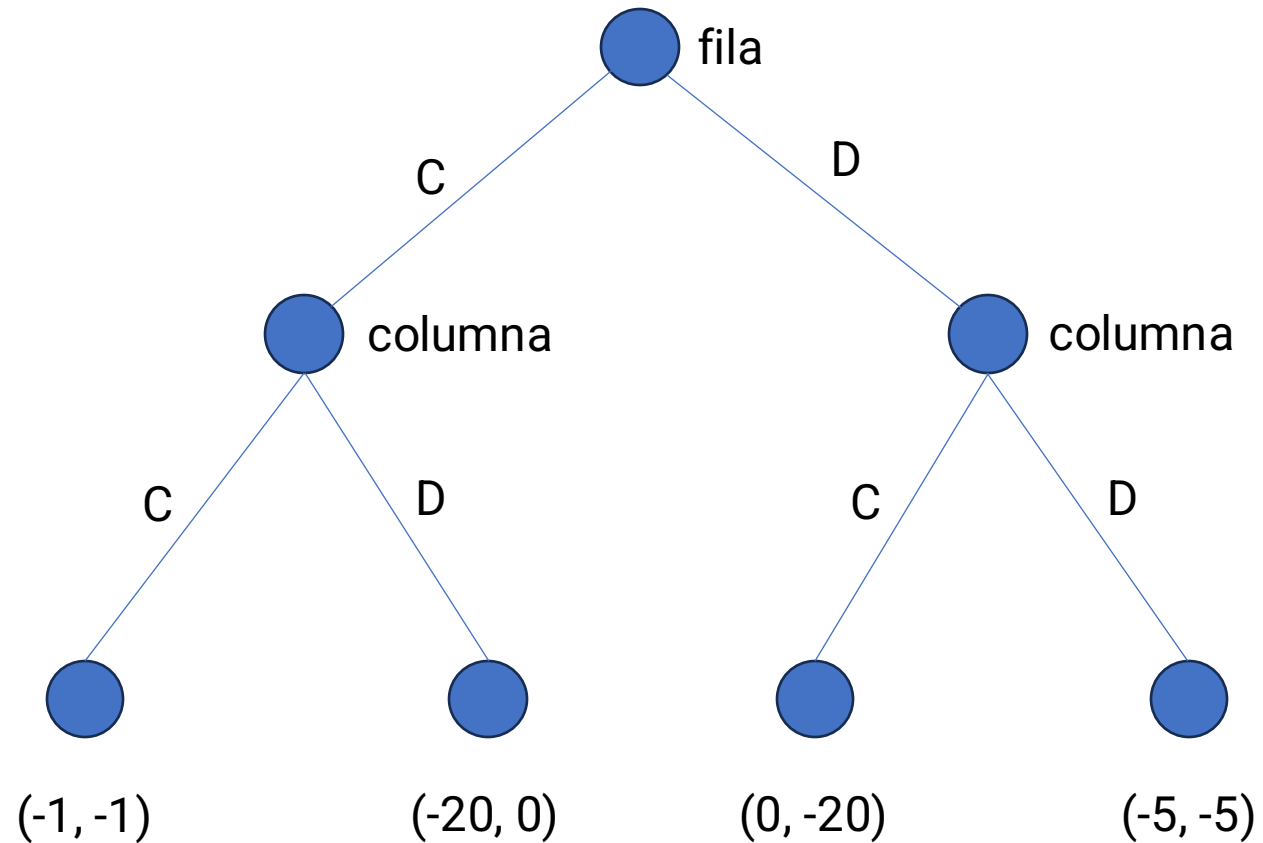
agente 1 agente 2	agente 2	
	yes	no
2-0	2, 0	0, 0
1-1	1, 1	0, 0
0-2	0, 2	0, 0

Ejemplo: *sharing game*



Multiagent systems: algorithmic, game-theoretic, and logical foundations
(Shoham & Leyton-Brown), Fig 5.1

Ejemplo: dilema del prisionero



Reducción a forma normal

- Dado un juego G en forma extensiva con información perfecta tal que $G = \langle N, A, H, Z, \mathcal{X}, \rho, u \rangle$, el **conjunto de estrategias puras para el agente i** es el conjunto

$$\prod_{h \in H, \rho(h)=i} \mathcal{X}(h)$$

- Es decir, es el **producto cartesiano de las acciones que el agente puede tomar en cada estado en el que *le puede tocar jugar***
 - En forma normal, la decisión era simultánea por lo que no hacía falta tener en cuenta en qué estado está el agente
 - En forma extensiva, la estrategia depende de los posibles estados en los que puede estar

Reducción a forma normal

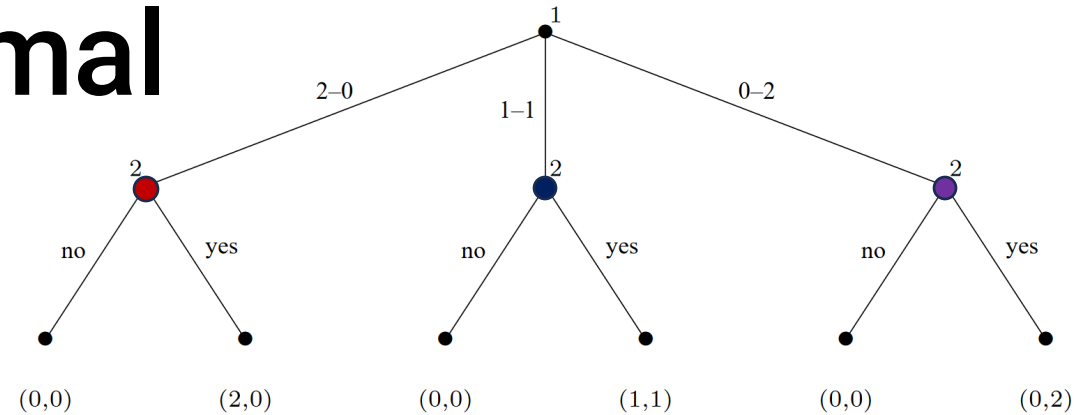
- En el *sharing game*, los conjuntos de estrategias son:
 - $S_1 = \{2 - 0, 1 - 1, 0 - 2\}$
 - $S_2 = \{(yes, yes, yes), (yes, yes, no), (yes, no, yes), (yes, no, no), (no, yes, yes), (no, yes, no), (no, no, yes), (no, no, no)\}$
- En el dilema del prisionero, los conjuntos de estrategias son:
 - $S_{fila} = \{C, D\}$
 - $S_{columna} = \{(C, C), (C, D), (D, C), (D, D)\}$
- Cada combinación $s_i \times s_j$ donde $s_i \in S_i, s_j \in S_j$ tiene un vector de recompensas asociado, por lo que **a partir de estos conjuntos de estrategias se puede construir la matriz de la forma normal**

Reducción a forma normal

- En el *sharing game*:

- $S_1 = \{2-0, 1-1, 0-2\}$

- $S_2 = \{(yes, yes, yes), (yes, yes, no), (yes, no, yes), (yes, no, no), (no, yes, yes), (no, yes, no), (no, no, yes), (no, no, no)\}$



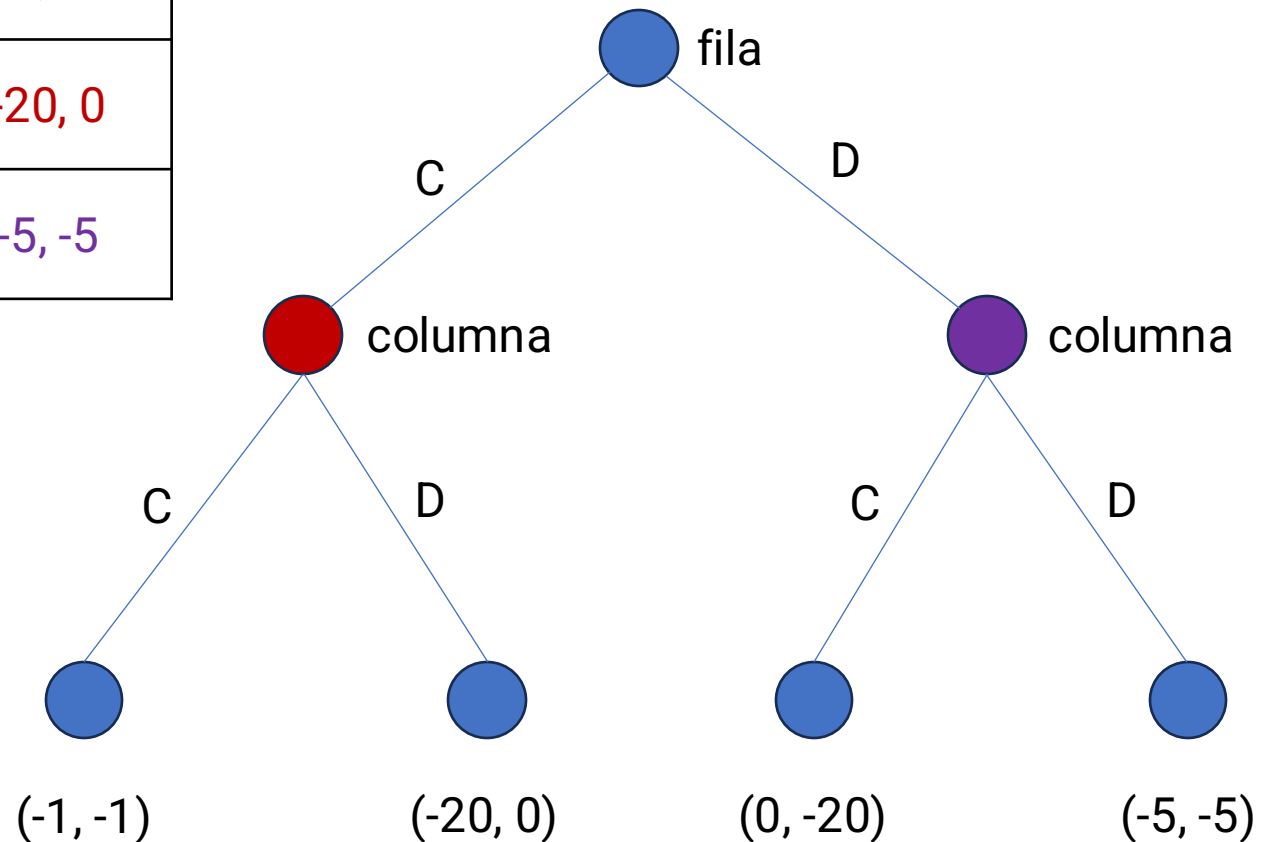
	yes, yes, yes	yes, yes, no	yes, no, yes	yes, no, no	no, yes, yes	no, yes, no	no, no, yes	no, no, no
2-0	2, 0	2, 0	2, 0	2, 0	0, 0	0, 0	0, 0	0, 0
1-1	1, 1	1, 1	0, 0	0, 0	1, 1	1, 1	0, 0	0, 0
0-2	0, 2	0, 0	0, 2	0, 0	0, 2	0, 0	0, 2	0, 0

Reducción a forma normal

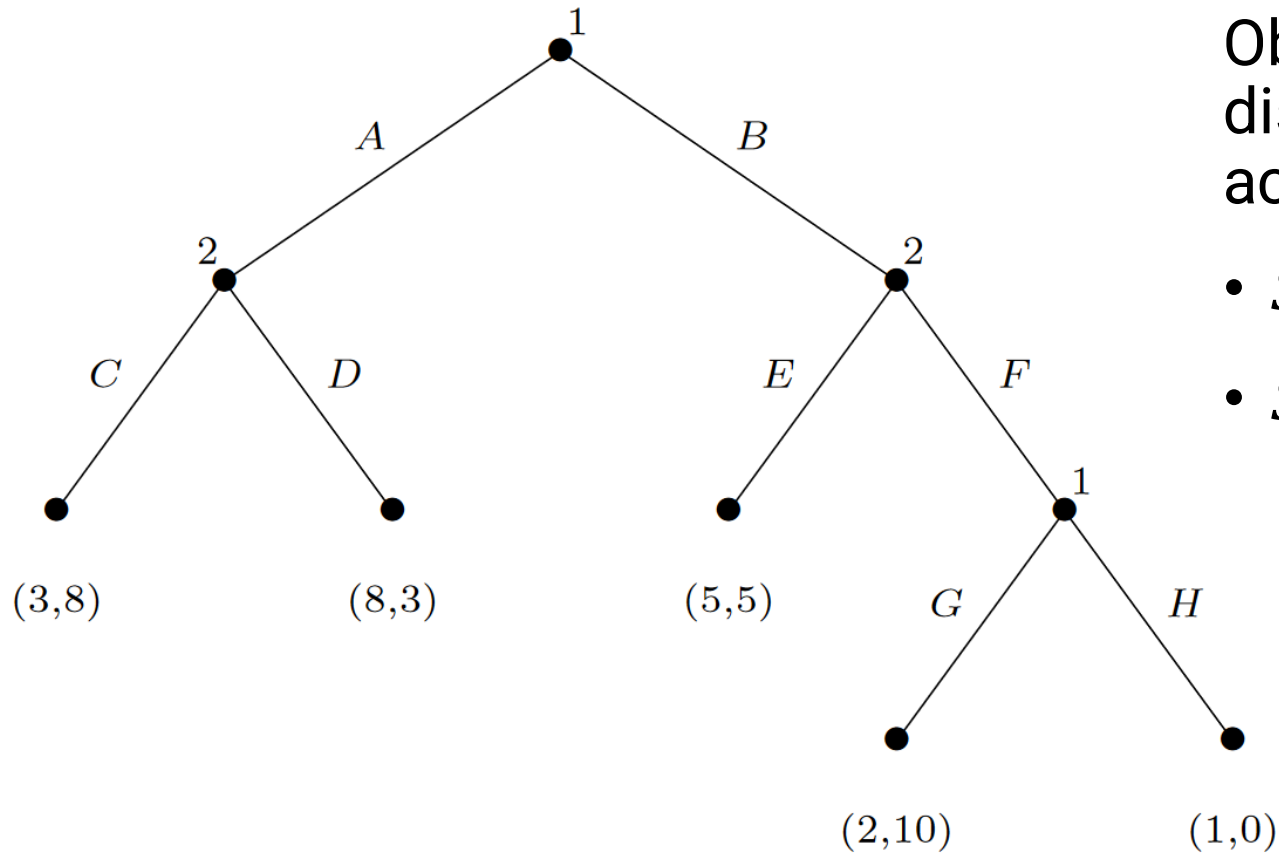
	C, C	C, D	D, C	D, D
C	-1, -1	-1, -1	-20, 0	-20, 0
D	0, -20	-5, -5	0, -20	-5, -5

En el dilema del prisionero:

- $S_{fila} = \{C, D\}$
- $S_{columna} = \{(C, C), (C, D), (D, C), (D, D)\}$



Reducción a forma normal

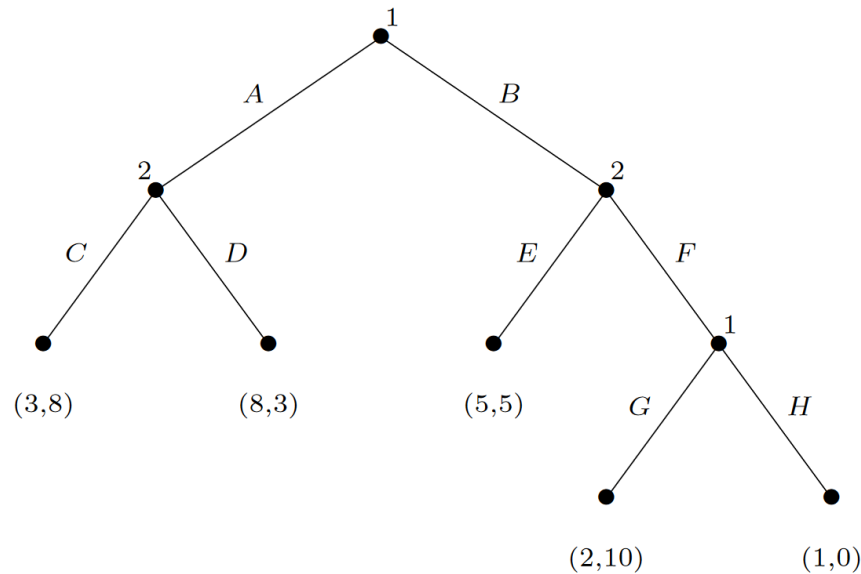


Observad que la definición no distingue entre los estados que son accesibles y los que no:

- $S_1 = \{(A, G), (A, H), (B, G), (B, H)\}$
- $S_2 = \{(C, E), (C, F), (D, E), (D, F)\}$

Reducción a forma normal

- $S_1 = \{(A, G), (A, H), (B, G), (B, H)\}$
- $S_2 = \{(C, E), (C, F), (D, E), (D, F)\}$

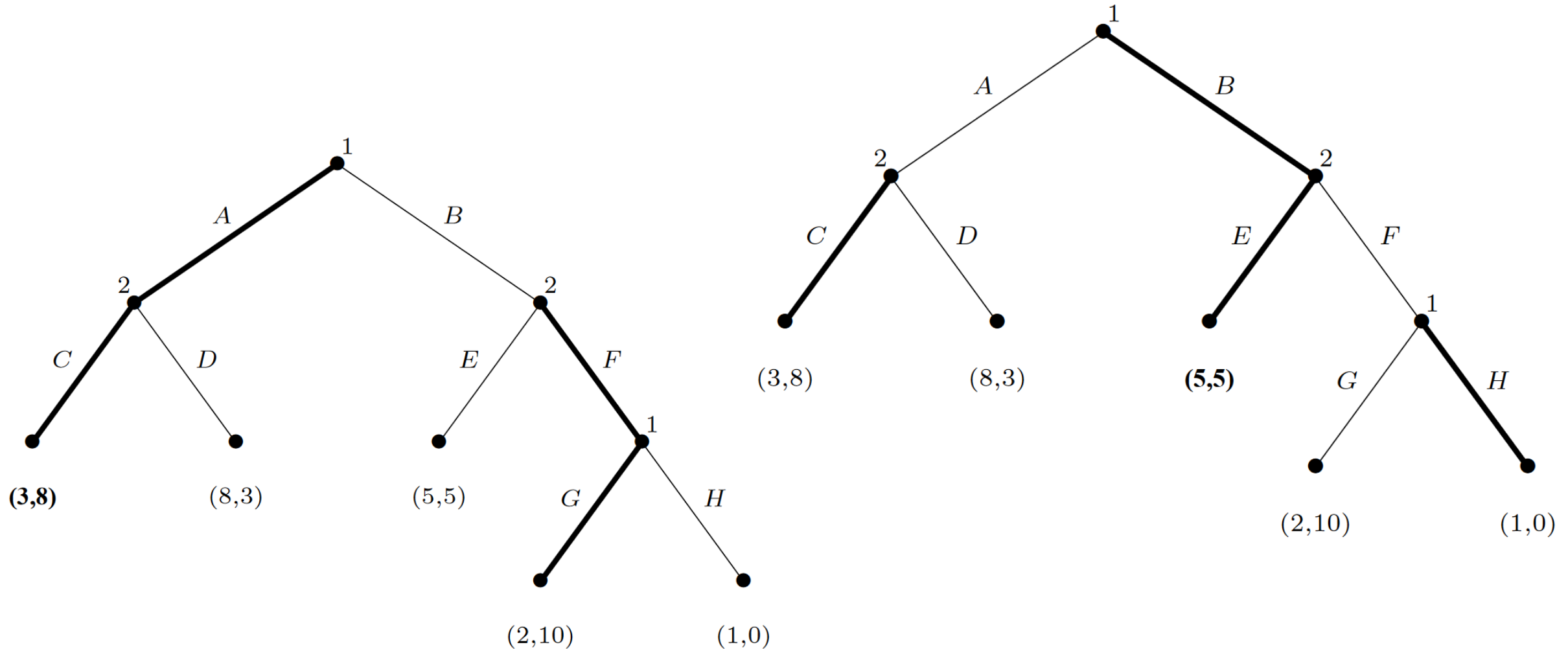


Multiagent systems: algorithmic, game-theoretic, and logical foundations (Shoham & Leyton-Brown), Fig 5.2

	(C,E)	(C,F)	(D,E)	(D,F)
(A,G)	3, 8	3, 8	8, 3	8, 3
(A,H)	3, 8	3, 8	8, 3	8, 3
(B,G)	5, 5	2, 10	5, 5	2, 10
(B,H)	5, 5	1, 0	5, 5	1, 0

Multiagent systems: algorithmic, game-theoretic, and logical foundations (Shoham & Leyton-Brown), Fig 5.3

Posibles equilibrios de Nash (ejemplo)

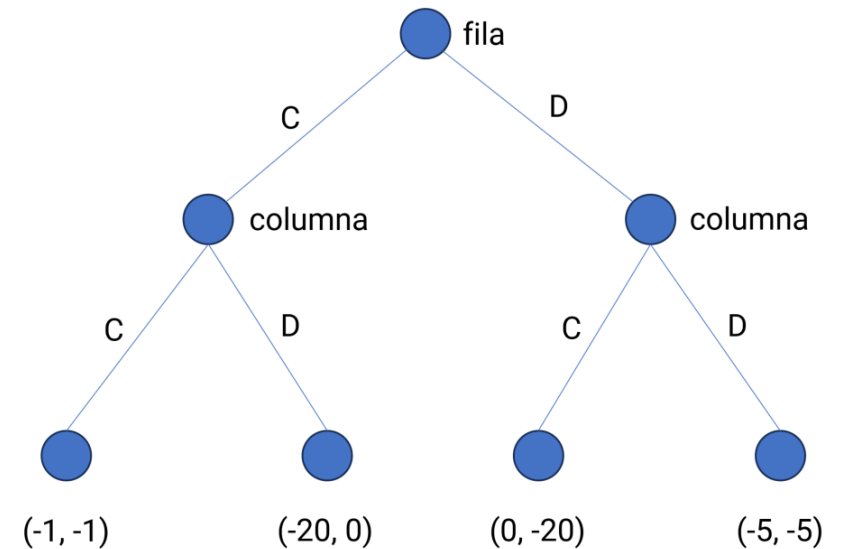
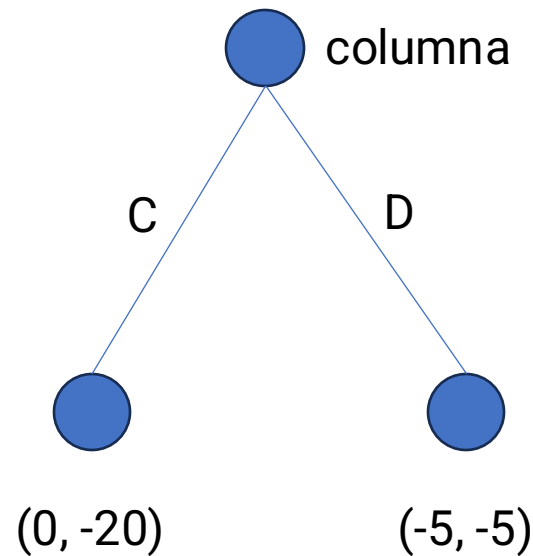
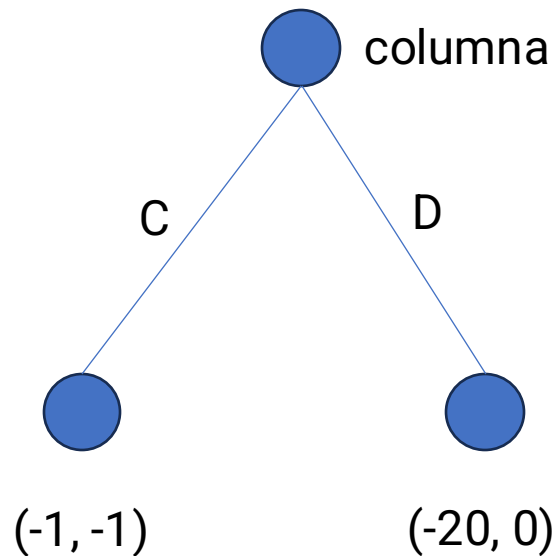


Equilibrio de Nash

- Todos los juegos en forma extensiva de información perfecta tienen un **equilibrio de Nash de estrategias puras**
 - No veremos la demostración, pero intuitivamente: al haber información perfecta y tener acceso al histórico antes de decidir cualquier acción, no es necesaria la introducción de distribuciones de probabilidad para alcanzar un equilibrio
- **El algoritmo de inducción hacia atrás permite encontrar *un* equilibrio de Nash**

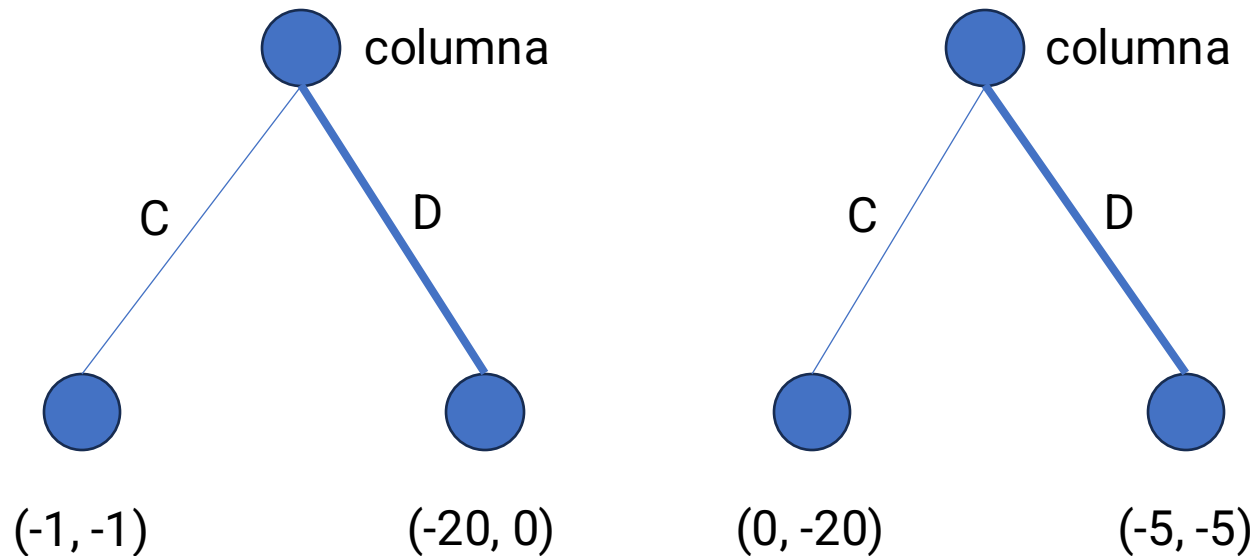
Equilibrio de Nash: definiciones

- Dado un juego en forma extensiva de información perfecta G , **el subjuego de G con raíz en el nodo h** es la restricción del juego G a los descendientes (directos e indirectos) de $h \in H$
- Ejemplo: subjuegos del dilema del prisionero



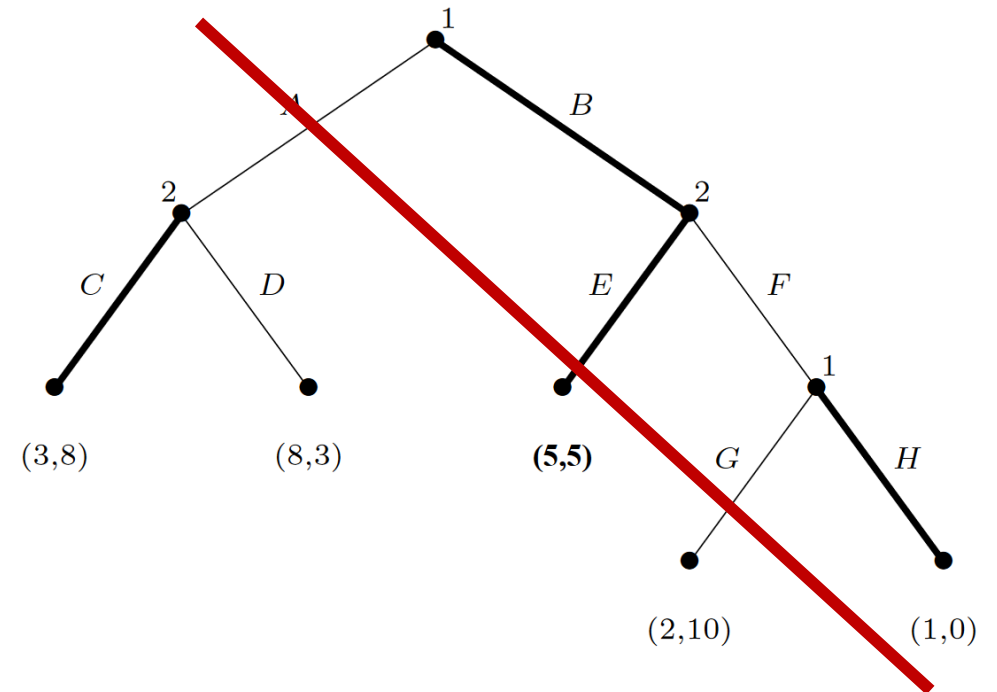
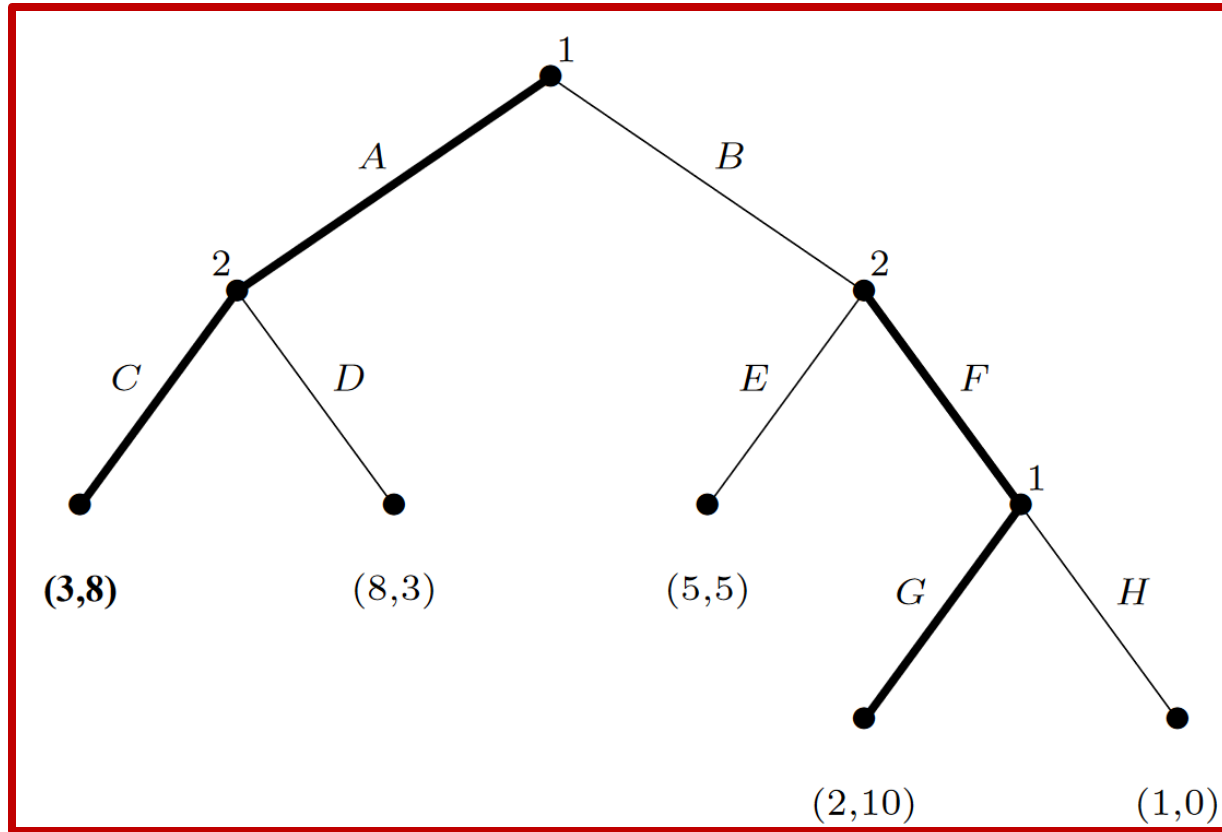
Equilibrio de Nash: definiciones

- Los **equilibrios perfectos (subgame-perfect equilibria, SPE)** para un **juego G** son el conjunto de todos los perfiles de estrategia s tal que para todo subjuego G' de G , la restricción de s para G' es un equilibrio de Nash para G'



- ~~Los **equilibrios perfectos (subgame-perfect equilibria, SPE)** para un **juego G** son el conjunto de todos los perfiles de estrategia s tal que para todo subjuego G' de G , la restricción de s para G' es un equilibrio de Nash para G'~~

Equilibrio perfecto de Nash



Inducción hacia atrás

función InducciónHaciaAtrás(nodo h) **retorna** $u(h)$: vector de recompensas para N

si $h \in Z$ **entonces**

retorna $u(h)$

$mejor_utilidad \leftarrow (-\infty, \dots, -\infty)$

para toda $a \in \mathcal{X}(h)$

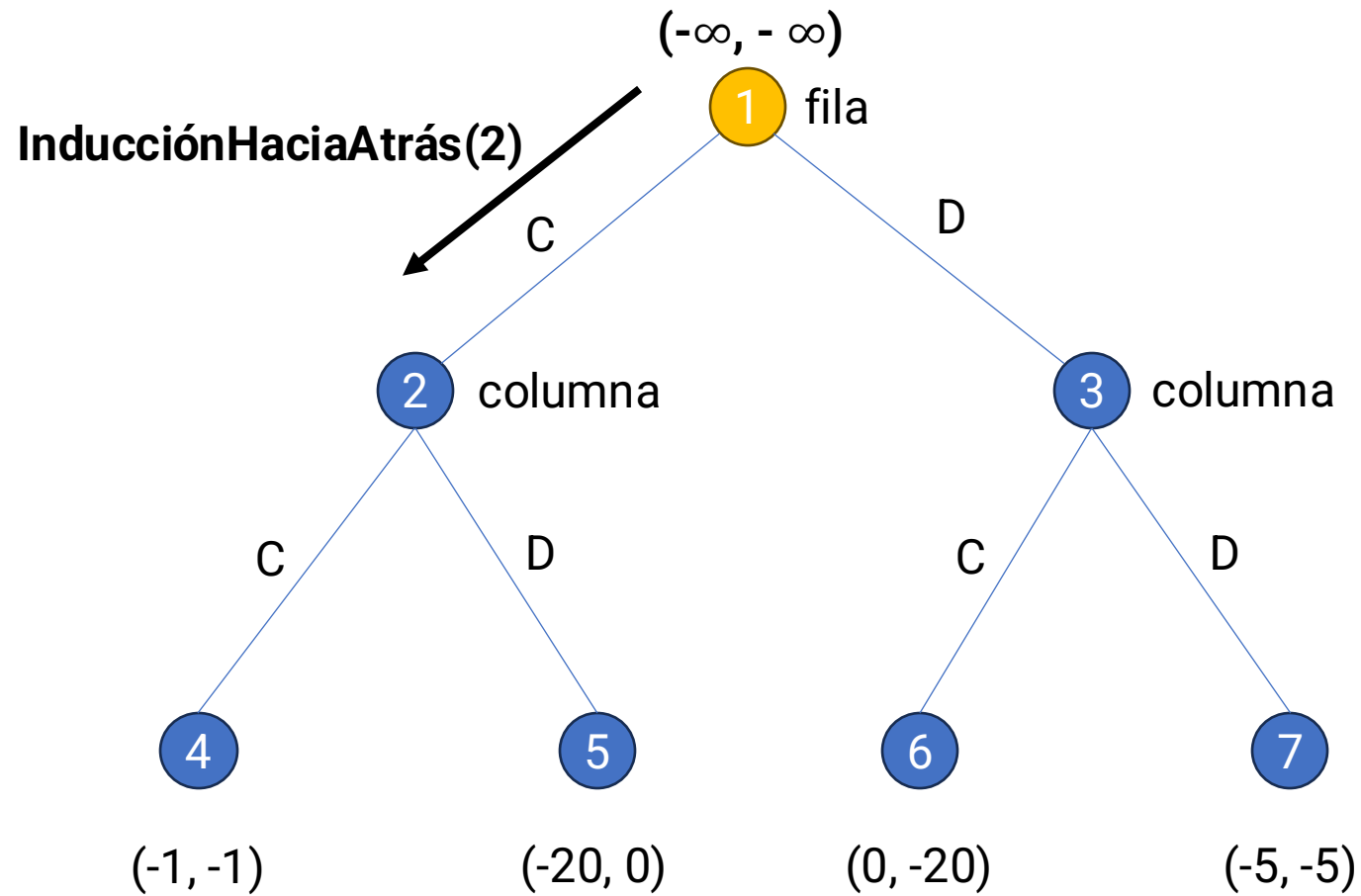
$utilidad_descendiente \leftarrow \text{InducciónHaciaAtrás}(\text{Sucesor}(h, a))$

si $utilidad_descendiente_{\rho(h)} > mejor_utilidad_{\rho(h)}$ **entonces**

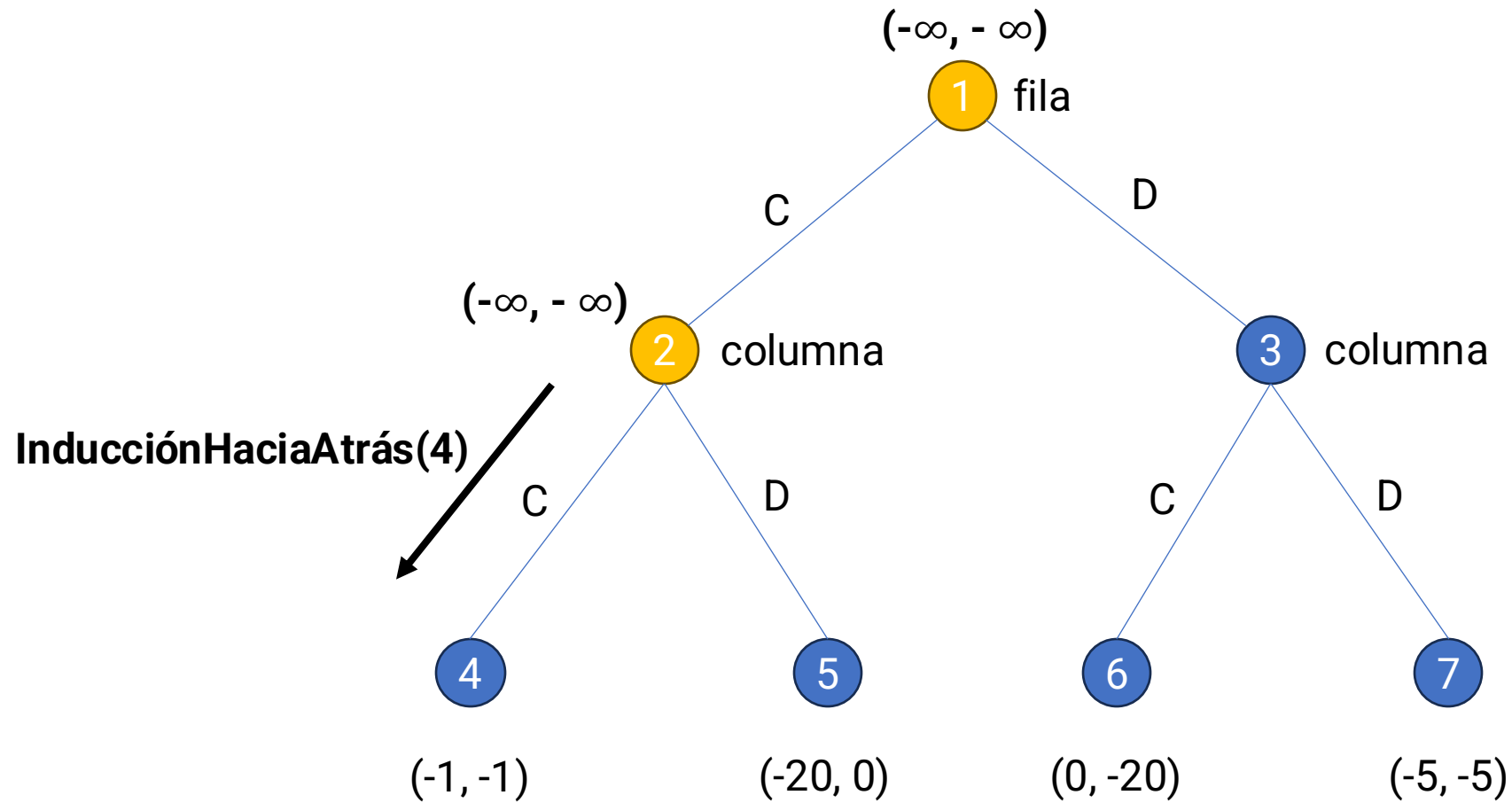
$mejor_utilidad \leftarrow utilidad_descendiente$

retornar $mejor_utilidad$

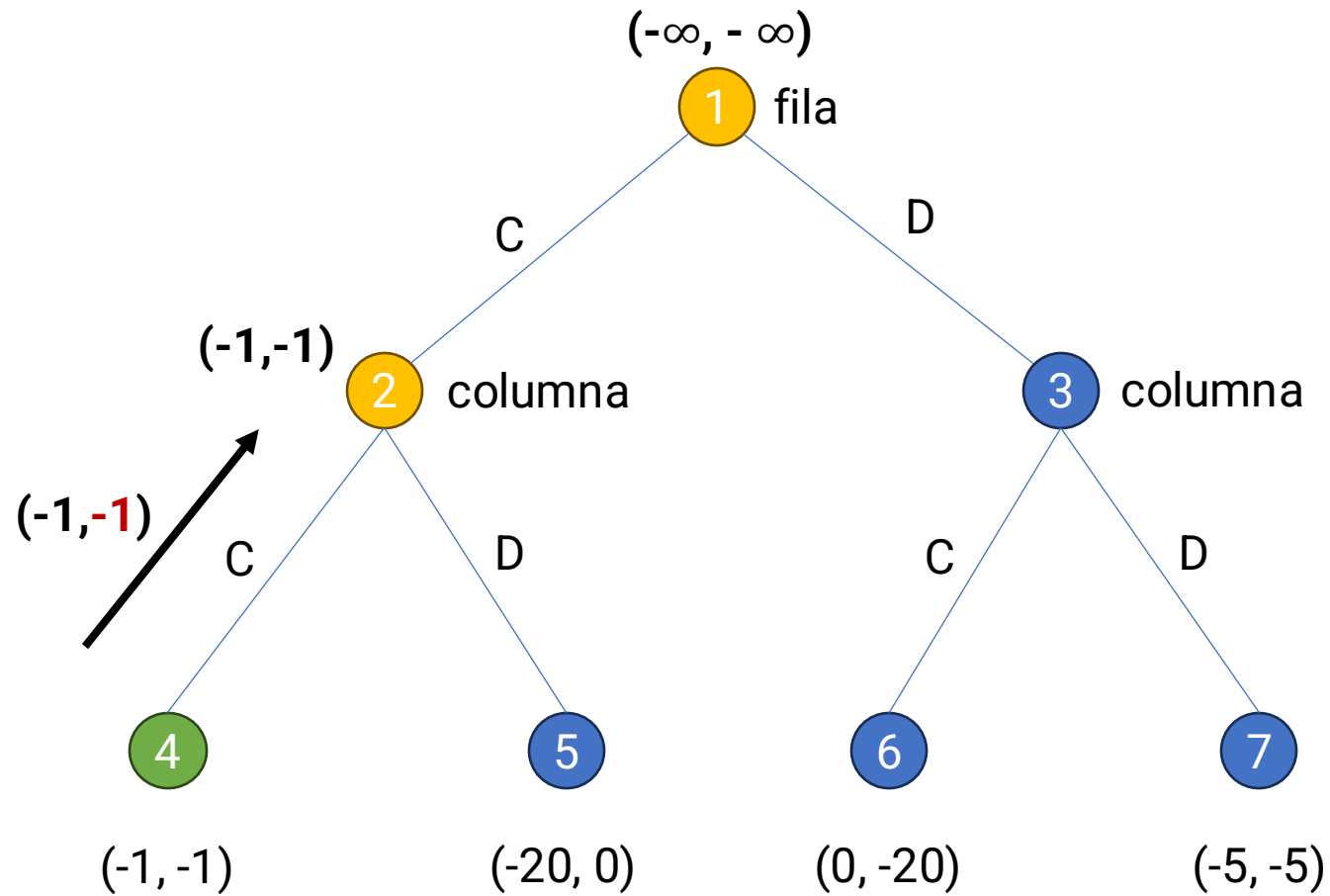
Inducción hacia atrás



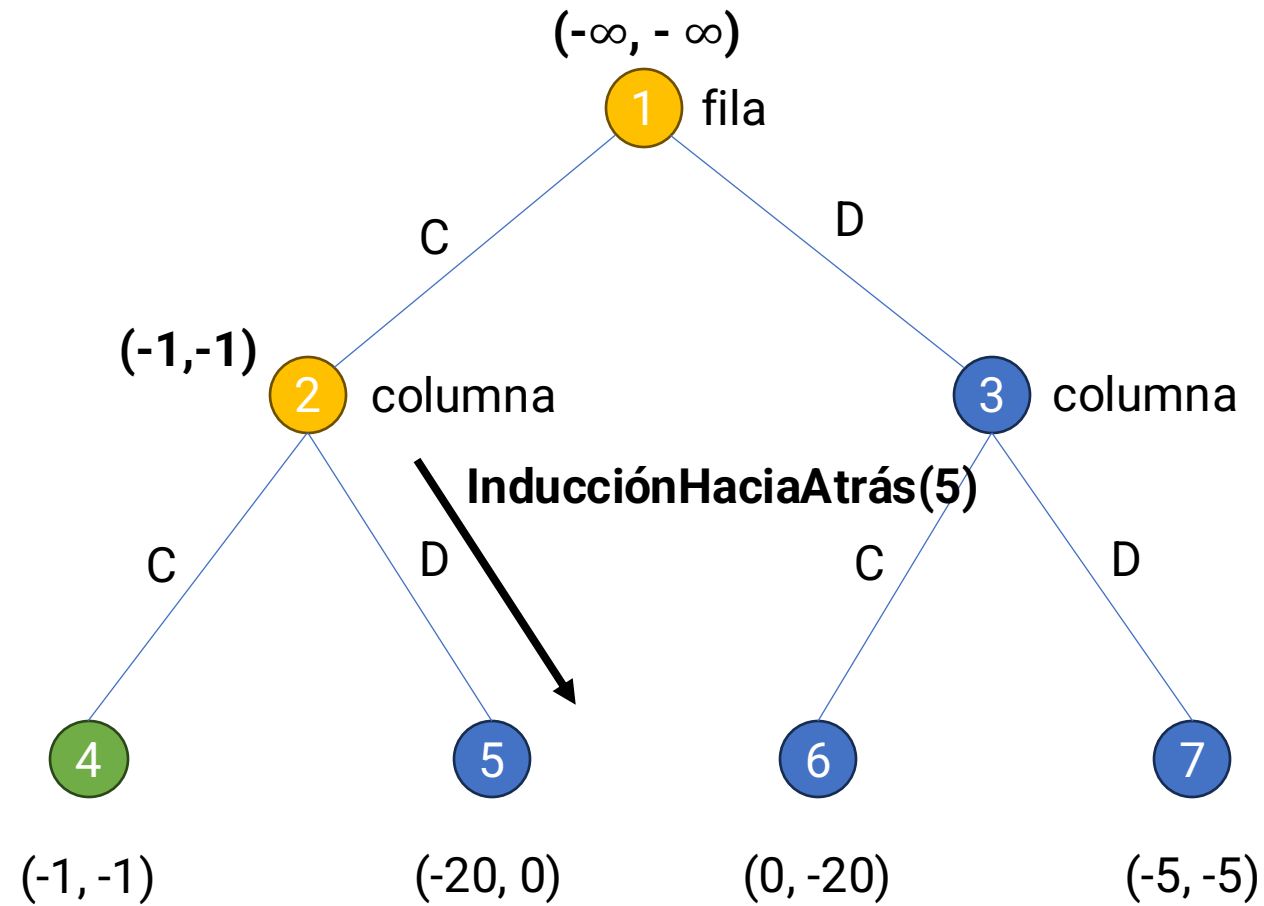
Inducción hacia atrás



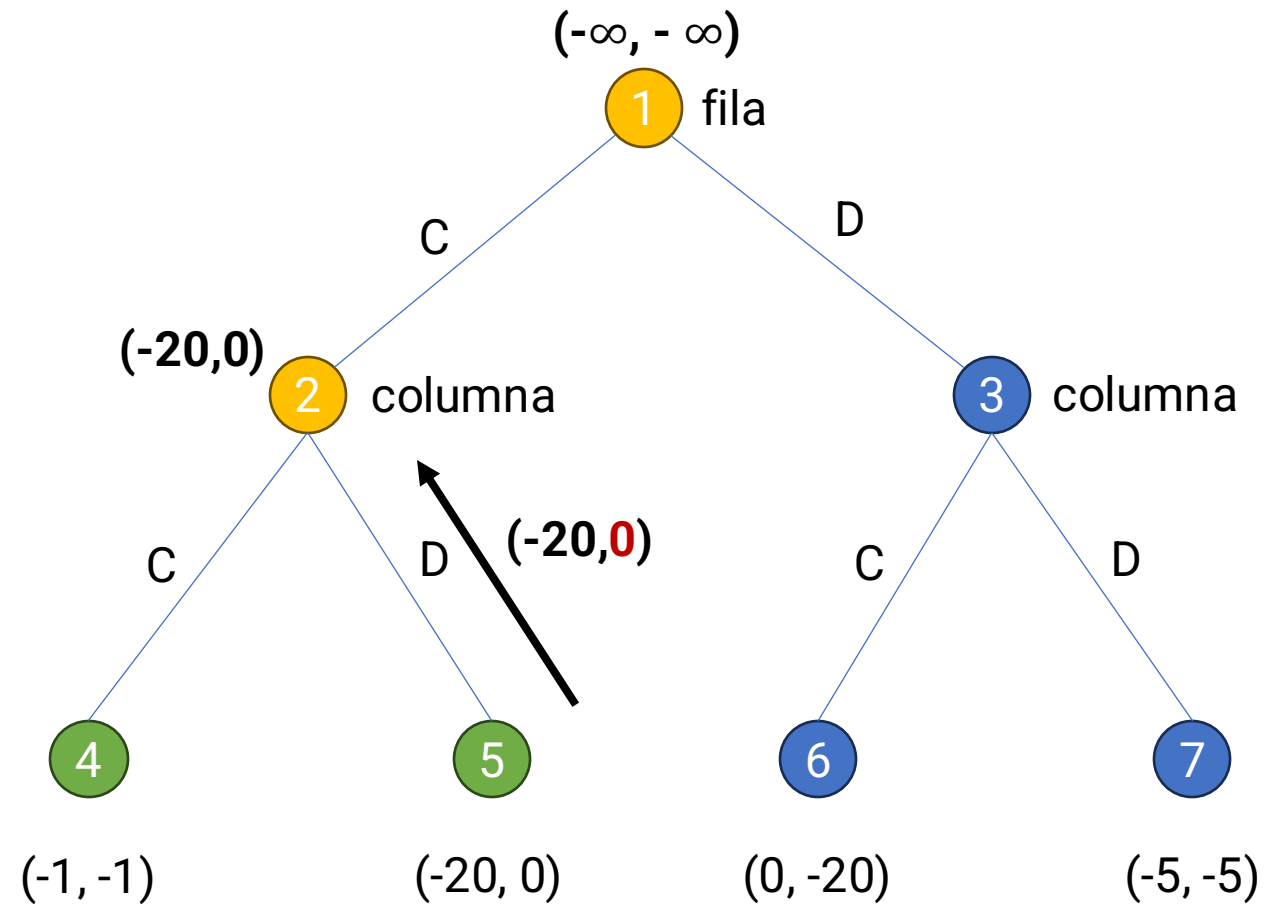
Inducción hacia atrás



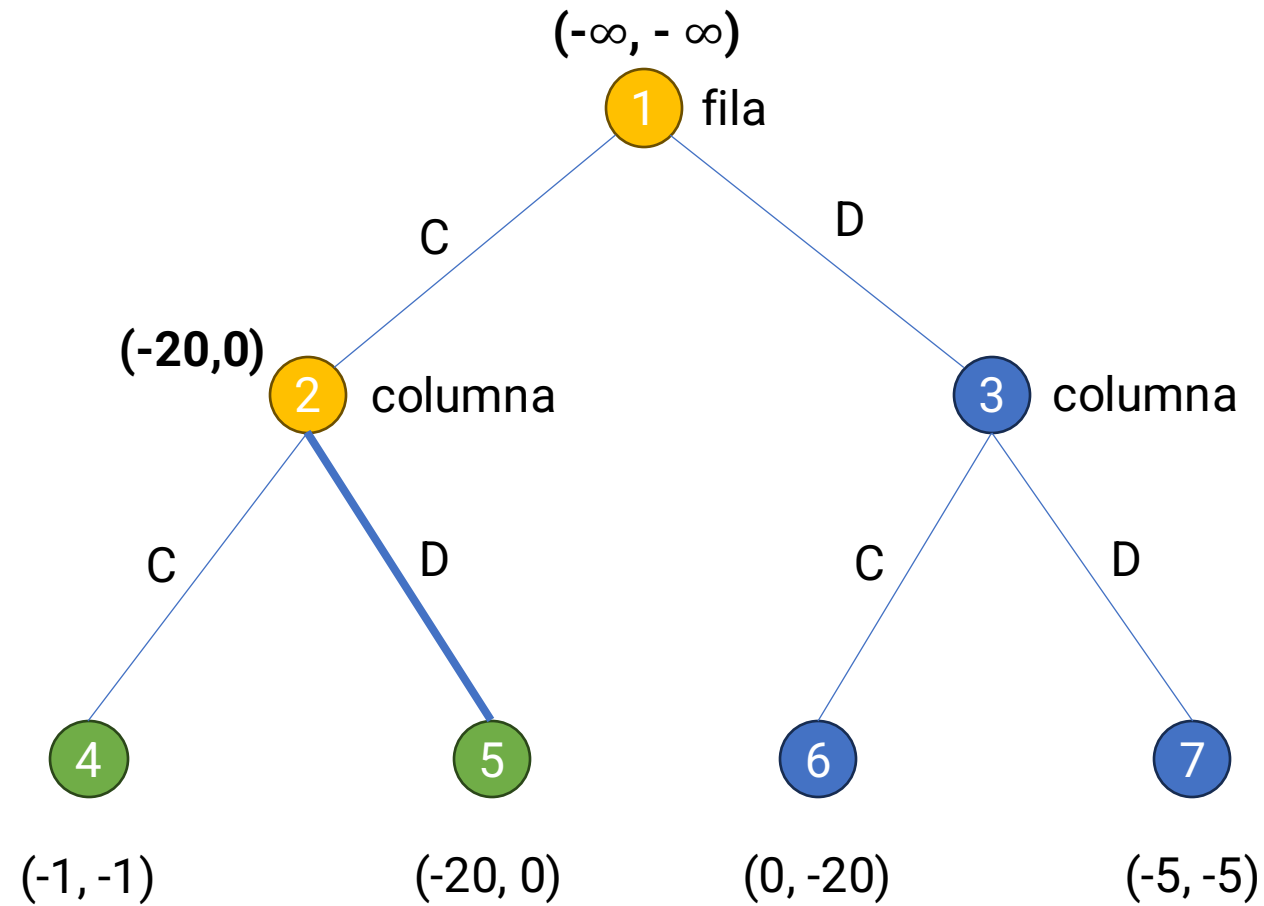
Inducción hacia atrás



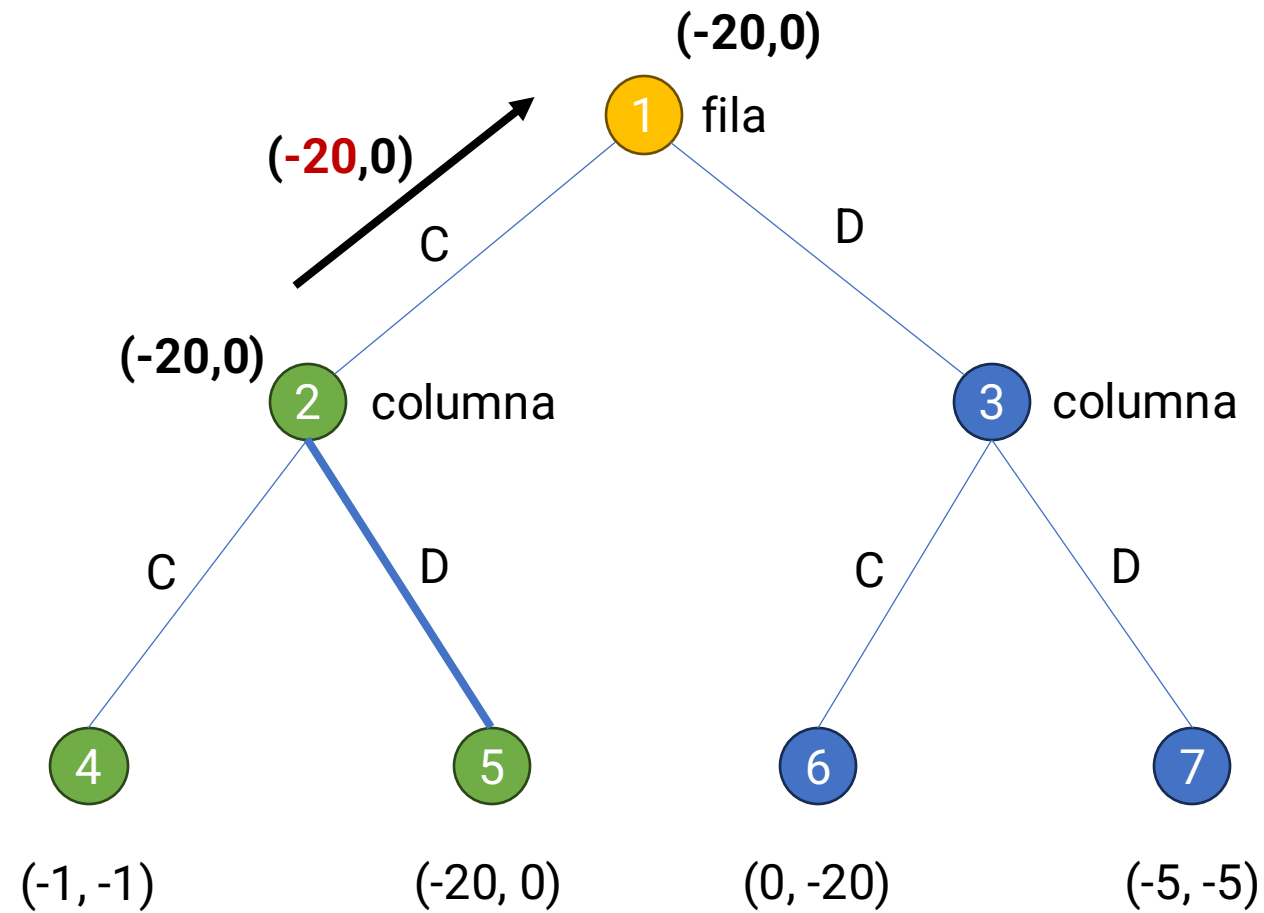
Inducción hacia atrás



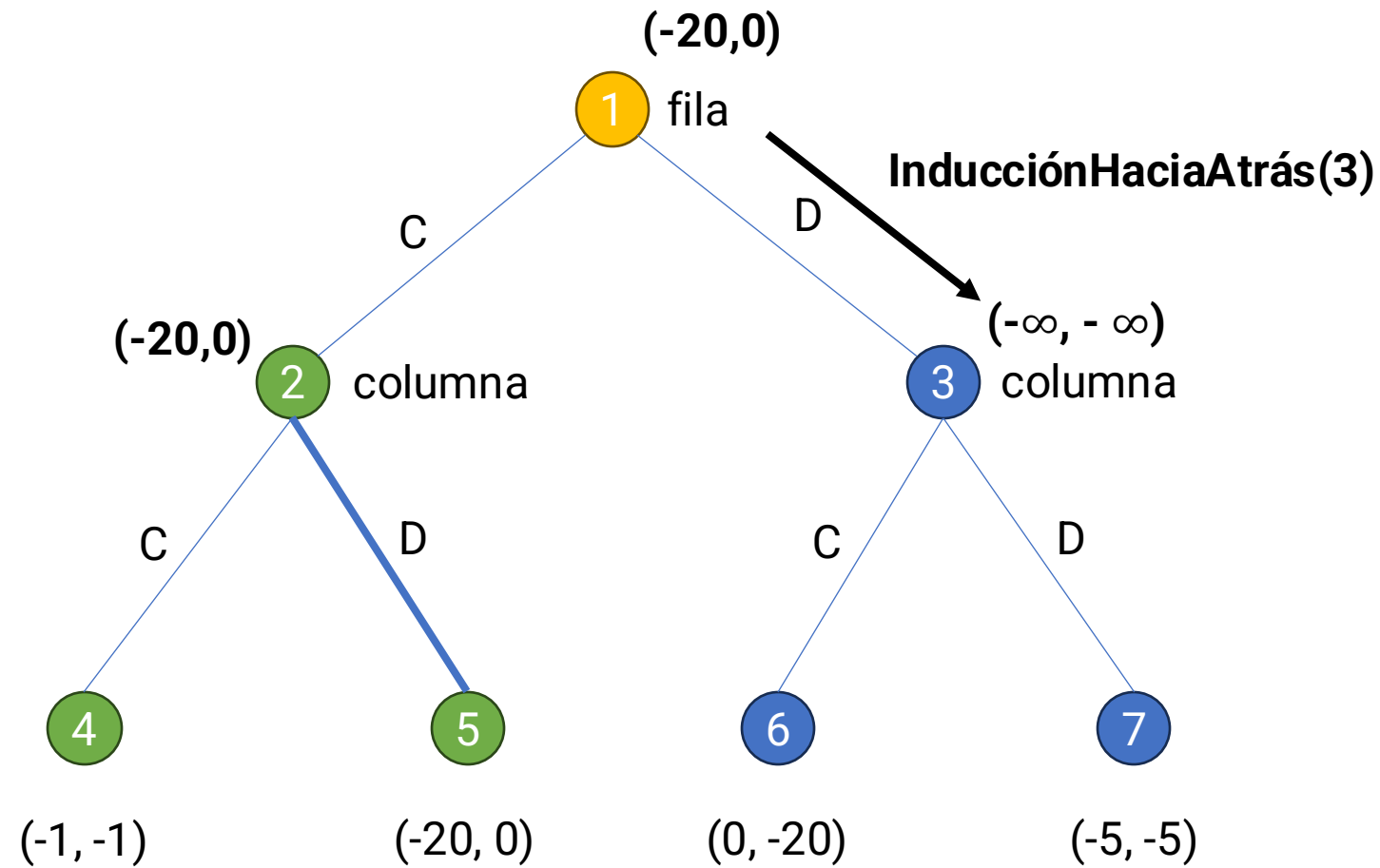
Inducción hacia atrás



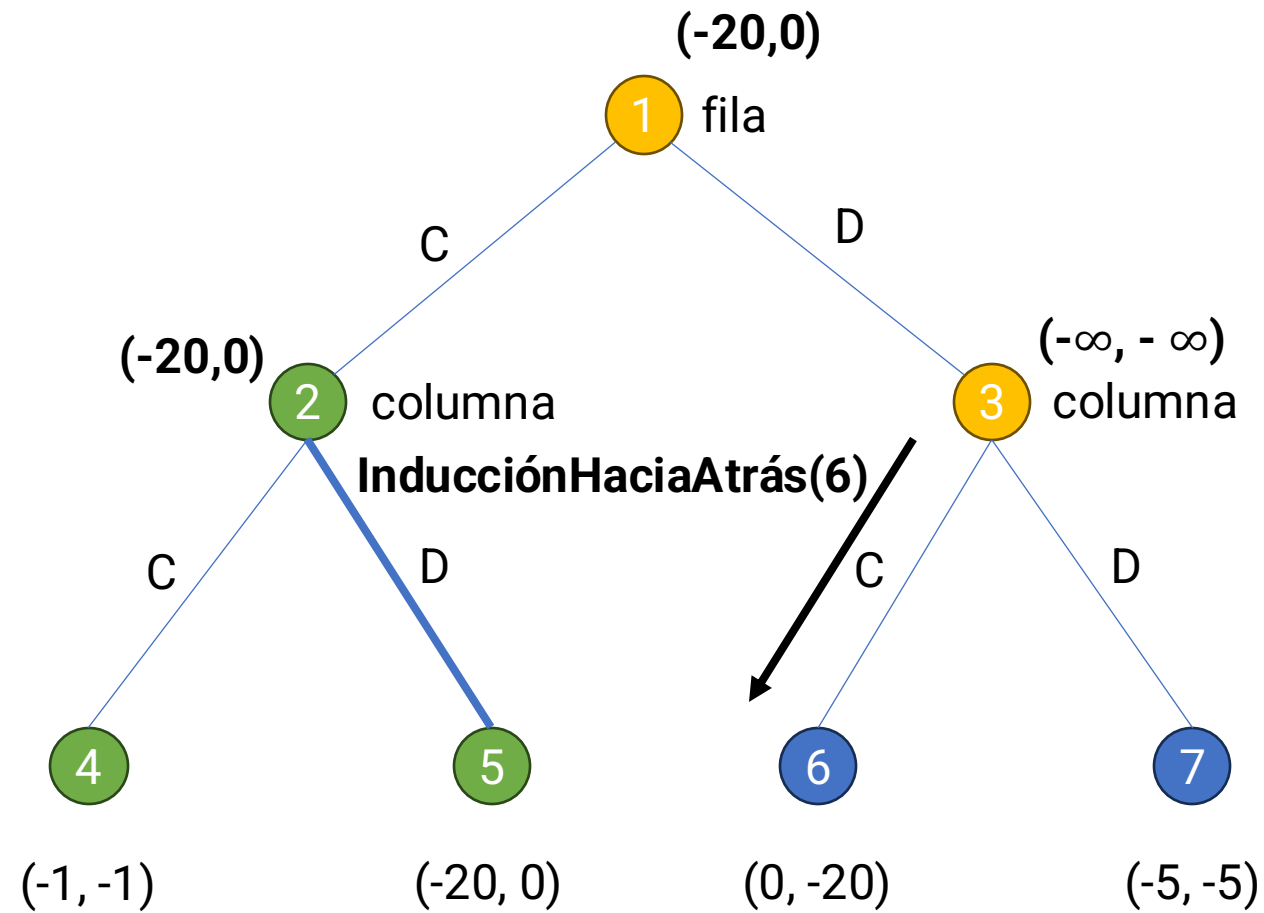
Inducción hacia atrás



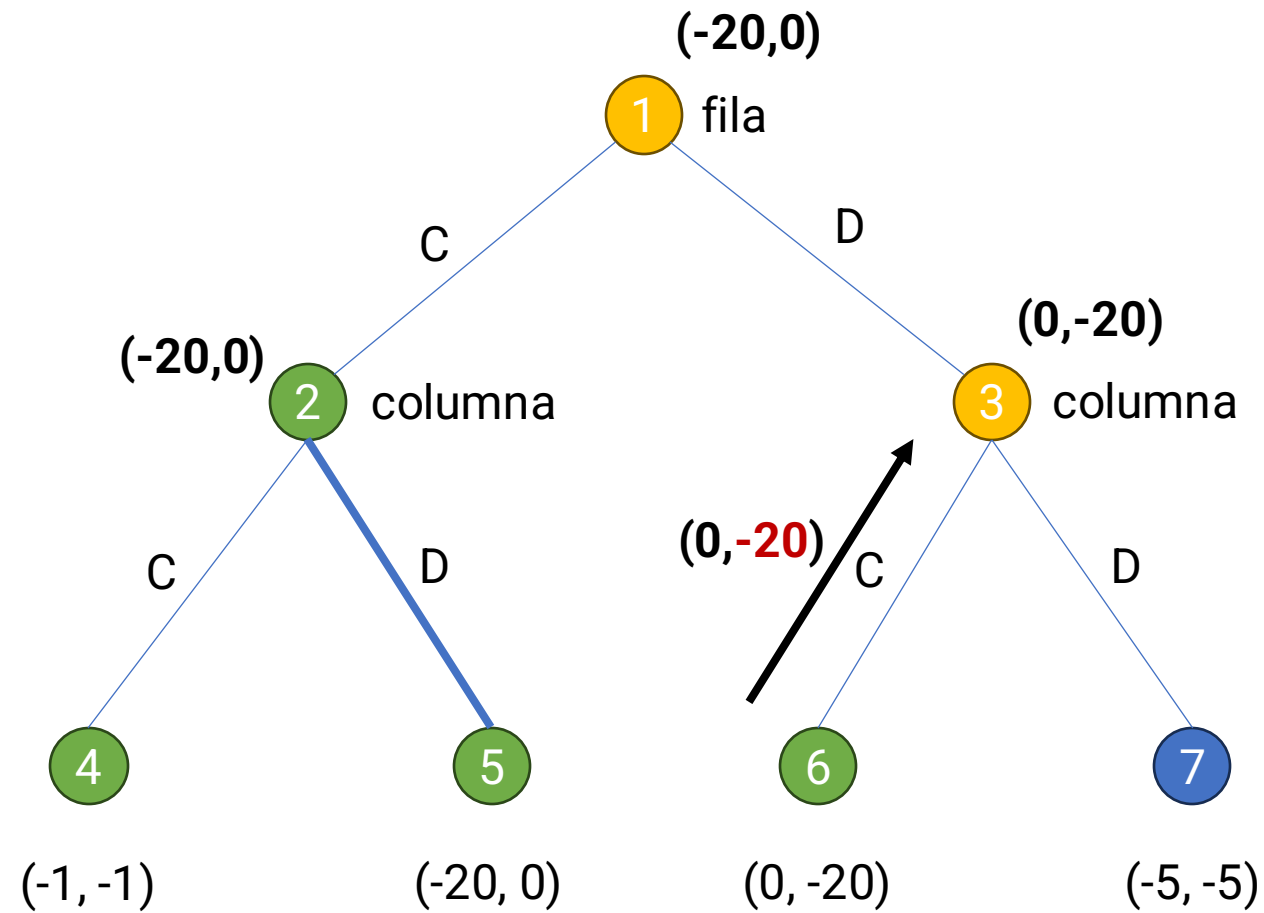
Inducción hacia atrás



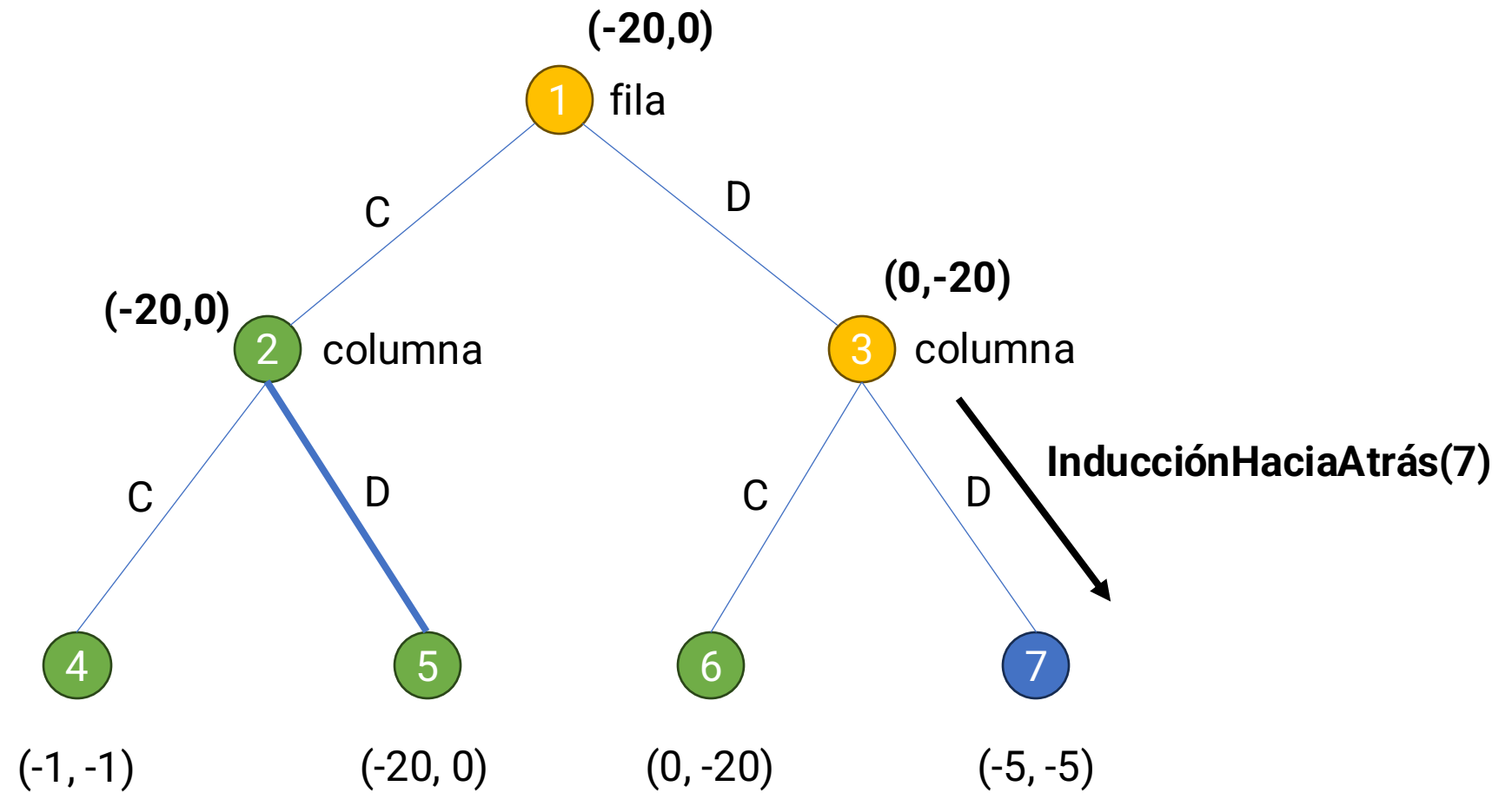
Inducción hacia atrás



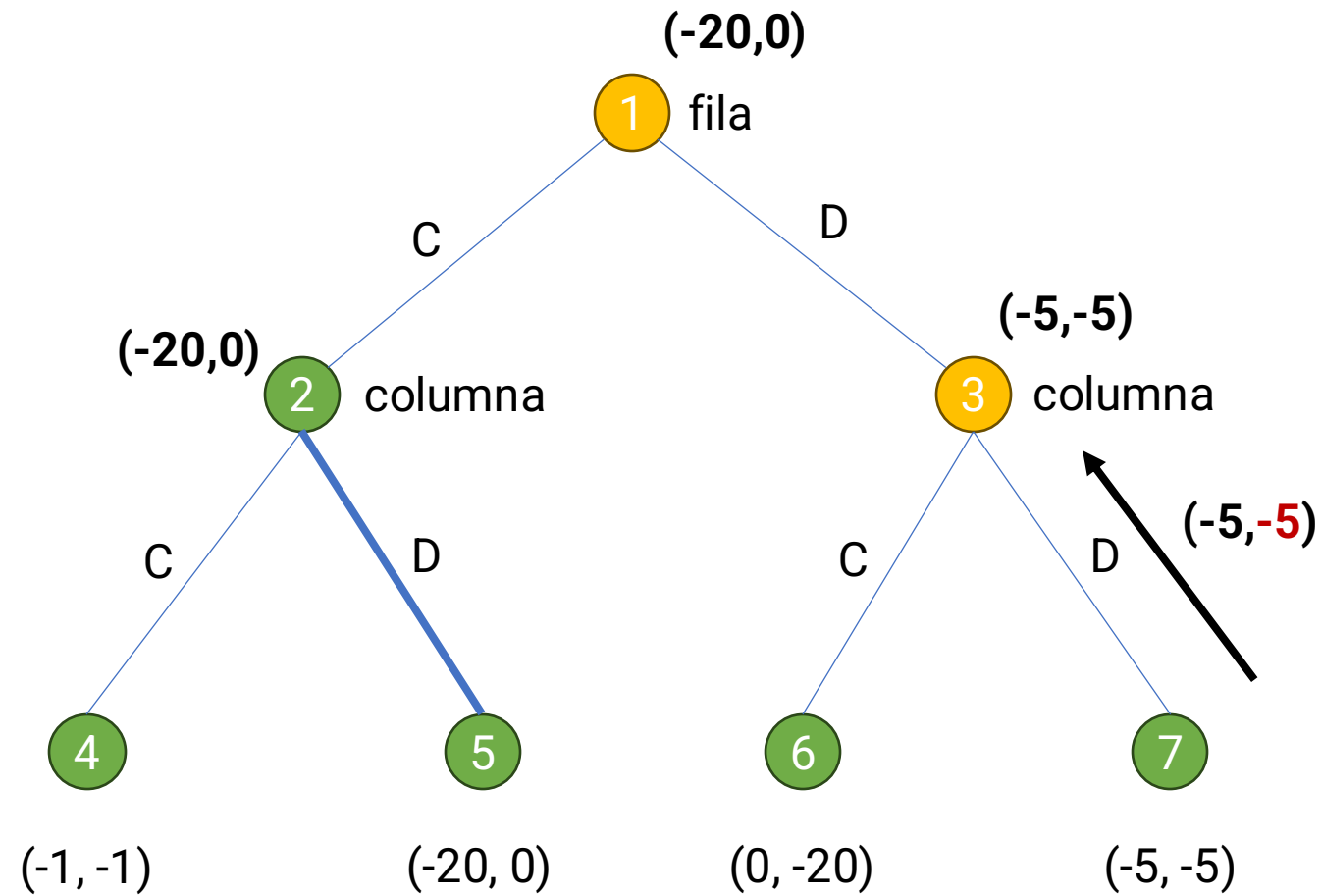
Inducción hacia atrás



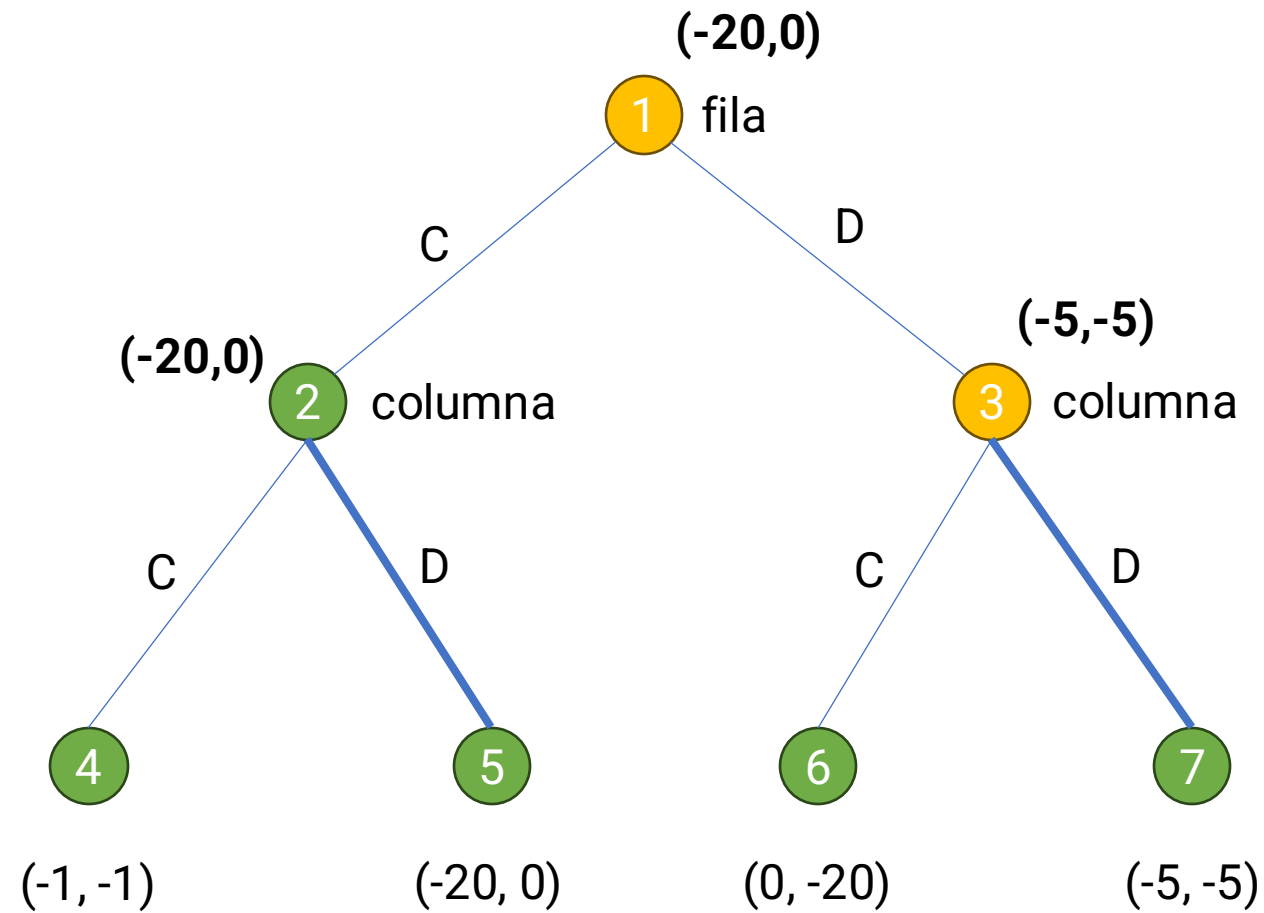
Inducción hacia atrás



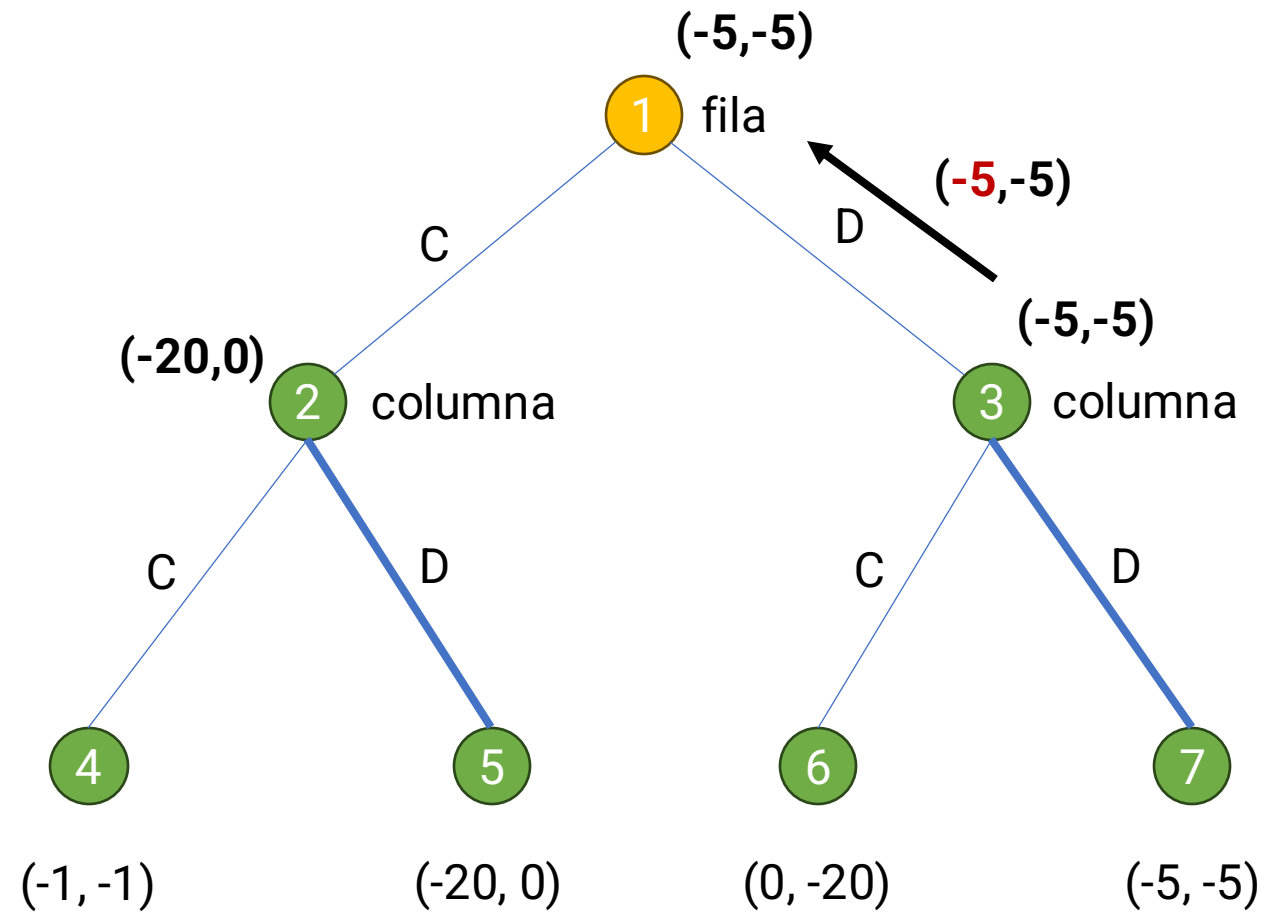
Inducción hacia atrás



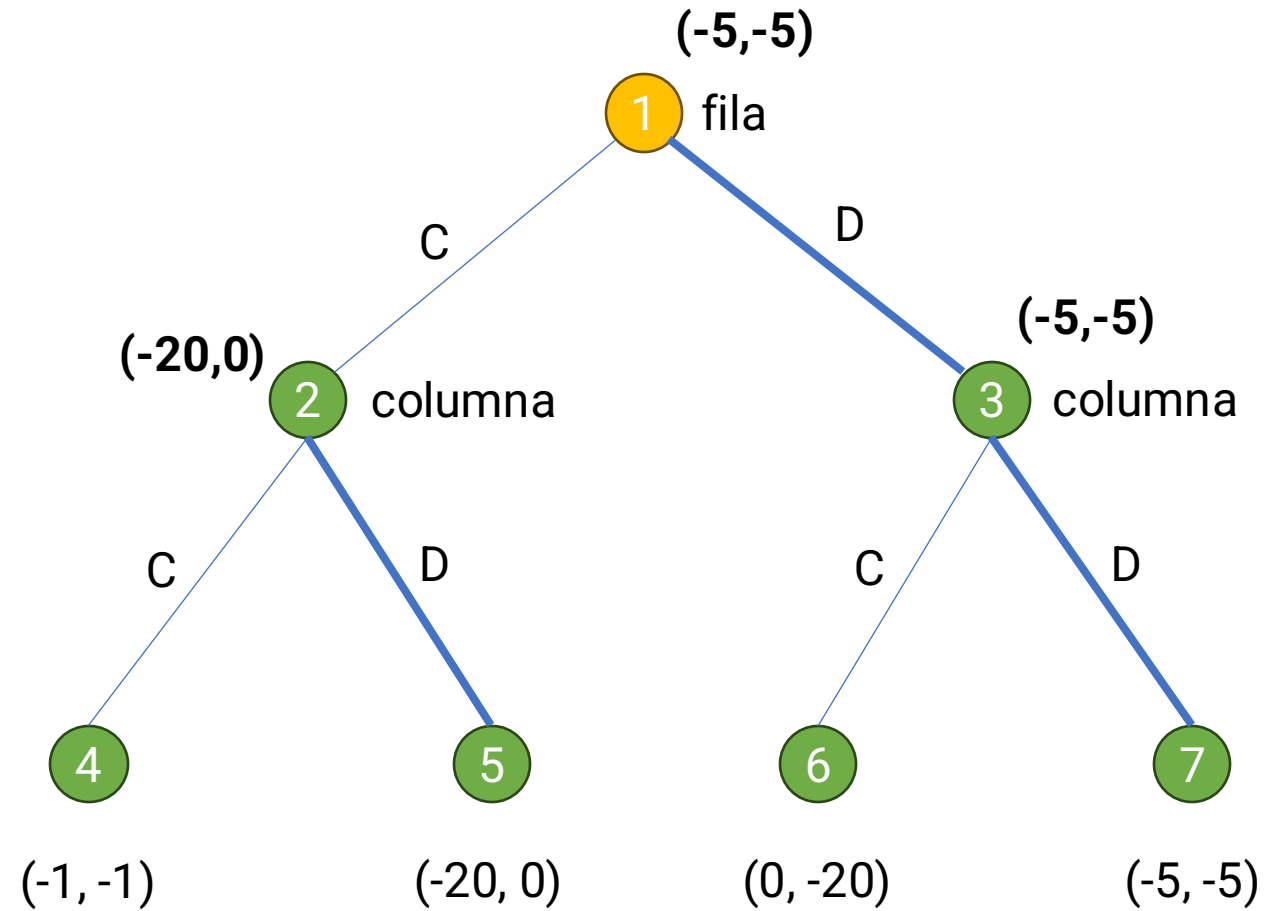
Inducción hacia atrás



Inducción hacia atrás



Inducción hacia atrás



Inducción hacia atrás

- Complejidad temporal: $O(b^m)$ para un factor de ramificación b y una profundidad máxima del árbol m
- Complejidad espacial: $O(b \cdot m)$
- Este algoritmo no permite encontrar todos los equilibrios de Nash, pero **sí garantiza encontrar al menos uno** para todos los juegos y subjuegos
 - En caso de empates entre las utilidades del agente que escoge, los equilibrios encontrados pueden ser diferentes debido al orden de visita al hacer la comparación:

$$utilidad_descendiente_{\rho(h)} > mejor_utilidad_{\rho(h)}$$

Negociación: regateo

Competición

Negociación como coordinación

- La negociación es el acto de "**resolver puntos de vista inconsistentes para llegar a un acuerdo**" (Lassri)
- Puede tener diversos objetos: costes, actividades, valor de verdad
- Es, en sí misma, **un proceso de coordinación**, ya que:
 - Los agentes acuerdan un conjunto predefinido de posibles acciones y reglas para el proceso de negociación
 - Tienen el objetivo común de llegar a un acuerdo
 - La información intercambiada a menudo contiene detalles de las medidas que deben adoptarse
- Sin embargo, es posible que los agentes no compartan exactamente el mismo objetivo dentro de la negociación: los compradores buscan un "precio" bajo, los vendedores buscan un "precio" alto

Métodos habituales

- **Enfoques basados en la Teoría de Juegos** (Nash, Levy, Zlotkin, Roschein): compartir funciones de utilidad o ver la convergencia de la negociación como la búsqueda de un equilibrio de Nash
- **Métodos recursivos e iterativos** (Lassri y otros): métodos / reglas de convergencia para negociaciones de varias rondas
- **Contract-Net (Iterativo)** (Simon and Davies): utilizando un mecanismo de llamada a ofertas y respuesta, en particular cuando se permiten contraofertas
- **Métodos basados en la argumentación** (Castelfranchi, Parsons, McBurney y otros): utilizando enunciados lógicos y juegos de diálogo para obligar a los agentes a llegar a un consenso

Problema de regateo

- Una transacción entre dos agentes puede formalizarse como un juego: **problema de regateo (*bargaining problem*)**
- Este formalismo permite analizar un problema competitivo desde una perspectiva “cooperativa” donde los dos agentes **buscan un acuerdo** que cumpla unas ciertas condiciones
- Dos agentes regatean por un determinado bien
 - Puede ser una asignación de tarea, acceso a una coalición, elección de un líder, etc.
 - Un agente comprador b tiene un precio de reserva p_b , tal que no aceptará un acuerdo por encima de p_b , y un valor para el bien v_b
 - Un agente vendedor s tiene un precio de reserva p_s , tal que no aceptará un acuerdo por debajo de p_s , y un valor para el bien v_s
 - Lo reducimos a utilidades: **las recompensas aceptables para el comprador cumplirán $u_b \geq v_b - p_b$, y para el vendedor cumplirán $u_s \geq p_s - v_s$**

Problema de regateo

- Un **problema de regateo** B se define como una tupla $\langle N, S, d \rangle$:
 - N es el conjunto de agentes
 - $S \subset \mathbb{R}^{|N|}$ es el **conjunto de posibles recompensas** que puede recibir cada agente, y a un conjunto $x \in S$ le llamamos alternativa
 - $d \in \mathbb{R}^{|N|}$ (*disagreement point*) es el **conjunto de recompensas por defecto**, es decir, la que recibiría cada agente en caso de no llegar a un acuerdo
 - Se han de cumplir las siguientes condiciones:
 - **Existe alguna solución que mejora la recompensa por defecto para todos los agentes:** $\exists x \in S: \forall i \in N: x_i > d_i$
 - **El conjunto S está acotado por las recompensas aceptables para cada agente según su precio de reserva y su valoración del acuerdo:** $\forall x \in S, i \in N: x_i \geq u_i$

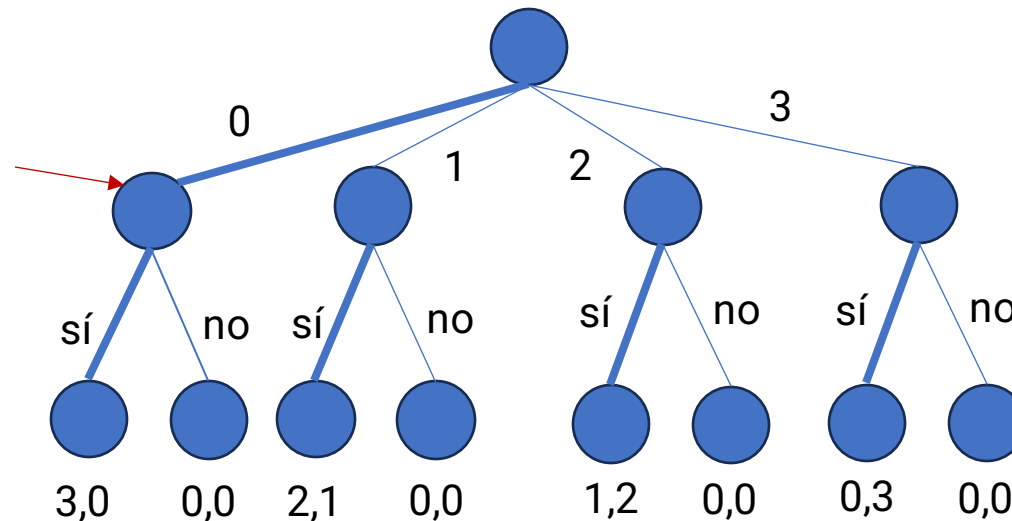
Regateo e inducción hacia atrás

- Un problema de regateo es reducible a un **juego en forma extensiva**
 - Las acciones pueden ser: ofertar, aceptar oferta, rechazar oferta, realizar contraoferta
- Por lo tanto, **el método de inducción hacia atrás nos permite encontrar un equilibrio de Nash**
 - Inducción hacia atrás es un concepto de solución válido para un proceso de negociación, **si se puede modelar como un problema de regateo**
- El problema de regateo más simple es el *ultimatum game*

Ultimatum game

- Dos agentes quieren repartirse una cantidad c
 - El agente A ofrece a B una cantidad $x \leq c$
 - Si B acepta, el vector de recompensas es $(c - x, x)$
 - Si B no acepta, el vector de recompensas es $(0, 0)$

Si aplicamos inducción hacia atrás,
tenemos dos equilibrios en el
subjuego $x = 0$
dependiendo del orden de visita del
agente B



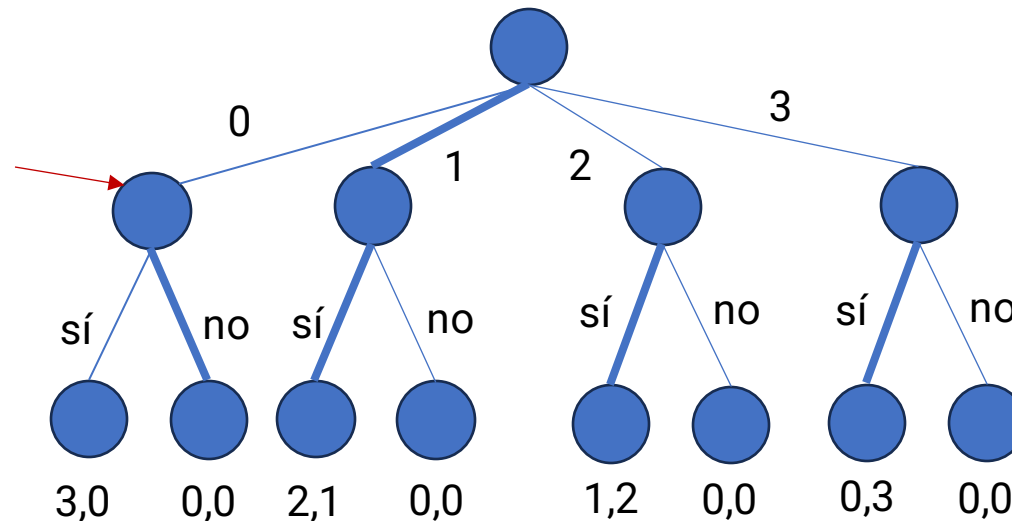
A

B

Ultimatum game

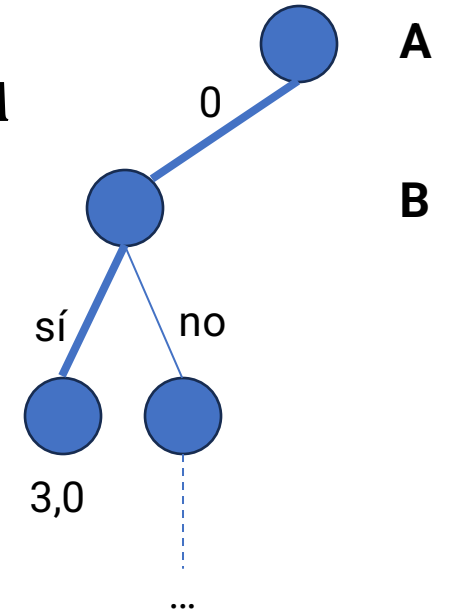
- Dos agentes quieren repartirse una cantidad c
 - El agente A ofrece a B una cantidad $x \leq c$
 - Si B acepta, el vector de recompensas es $(c - x, x)$
 - Si B no acepta, el vector de recompensas es $(0, 0)$

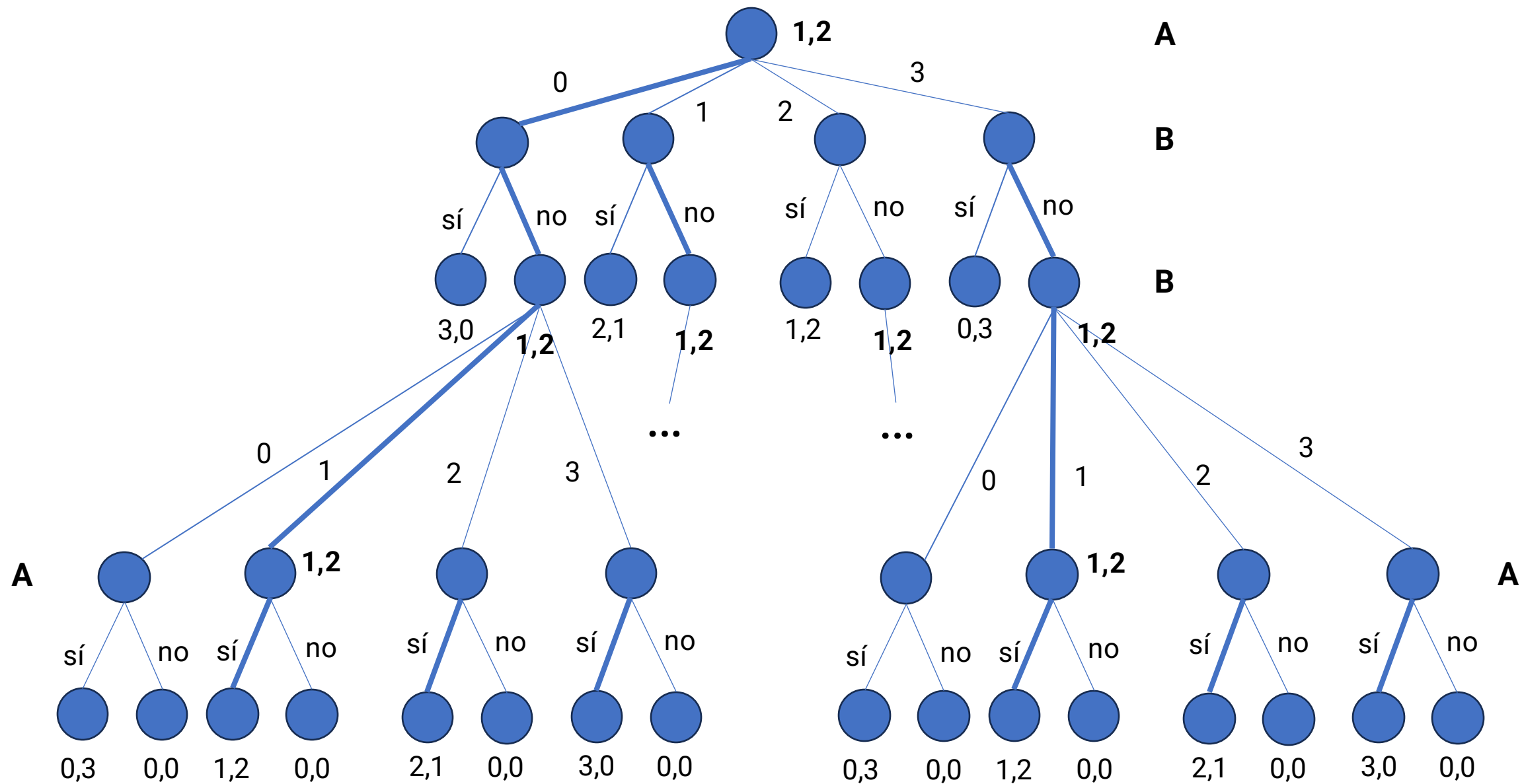
Si aplicamos inducción hacia atrás,
tenemos dos equilibrios en el
subjuego $x = 0$
dependiendo del orden de visita del
agente B



Ultimatum game: equilibrios

- Si $S \subset \mathbb{N}^2$, en el subjuego $x = 0$ no hay mejor respuesta de B (es indiferente), por lo que hay dos equilibrios en inducción hacia atrás:
 - $x = 0$, B acepta $x = 0$: $(3, 0)$
 - $x = 1$ (el valor positivo más pequeño), B acepta $x = 1$: $(2, 1)$
- Sin embargo, si $S \subset \mathbb{R}^2$, **cualquier oferta $x > 0$ es subóptima para A** ya que siempre puede haber una oferta $x' < x$
 - El único equilibrio en este caso es $x = 0$, B acepta: $(3, 0)$
- **Este juego se puede extender**, permitiendo a B realizar una contraoferta si su respuesta es no aceptar:
 - El agente B ofrece a A una cantidad $y \leq c$
 - Si A acepta, el vector de recompensas es $(y, c - y)$
 - Si A no acepta, el vector de recompensas es $(0, 0)$





Ultimatum game con horizonte finito

- Cada situación a partir de cada respuesta de no aceptar es un **subjuego equivalente al *ultimatum game* de una decisión**, cambiando A por B
- Supongamos un horizonte finito, con los jugadores turnándose
- Si la última oferta es de A , un equilibrio válido es:
 - Si $S \subseteq \mathbb{N}^2$: $x = 1$, B acepta: $(c - 1, 1)$
 - Si $S \subseteq \mathbb{R}^2$: $x = 0$, B acepta: $(c, 0)$
- Si la última oferta es de B , un equilibrio válido es:
 - Si $S \subseteq \mathbb{N}^2$: $y = 1$, A acepta: $(1, c - 1)$
 - Si $S \subseteq \mathbb{R}^2$: $y = 0$, A acepta: $(0, c)$

Ultimatum game con horizonte finito

- Por lo tanto, **el último agente en responder está siempre en desventaja**: no hay una manera justa de decidir cuántos turnos debería durar
- Una forma alternativa más realista es modelando un cierto grado de **impaciencia** en los agentes mediante un **factor de descuento**: $0 < \delta_i < 1$
- Las recompensas en cada turno en caso de acuerdo son:
 - Turno 1: A ofrece x_1 , y si acepta B , reciben $(1 - x_1, x_1)$
 - Turno 2: B ofrece y_1 , y si acepta A , reciben $(\delta_A y_1, \delta_B (1 - y_1))$
 - Turno 3: A ofrece x_2 , si acepta B , reciben $(\delta_A^2 (1 - x_2), \delta_B^2 x_2)$
 - Turno 4: B ofrece y_2 , si acepta A , reciben $(\delta_A^3 y_2, \delta_B^3 (1 - y_2))$
- La utilización de factores de descuento permite calcular equilibrios en juegos de regateo con horizonte infinito (aunque la inducción hacia atrás deja de ser aplicable)

Ultimatum game con horizonte finito

Por inducción hacia atrás, con $S \subset \mathbb{R}^2$ y suponiendo que el último agente en ofertar es B en el último turno ($T + 1$) suponiendo un reparto restante c :

Turno $T + 1$ (oferta de B y respuesta de A):

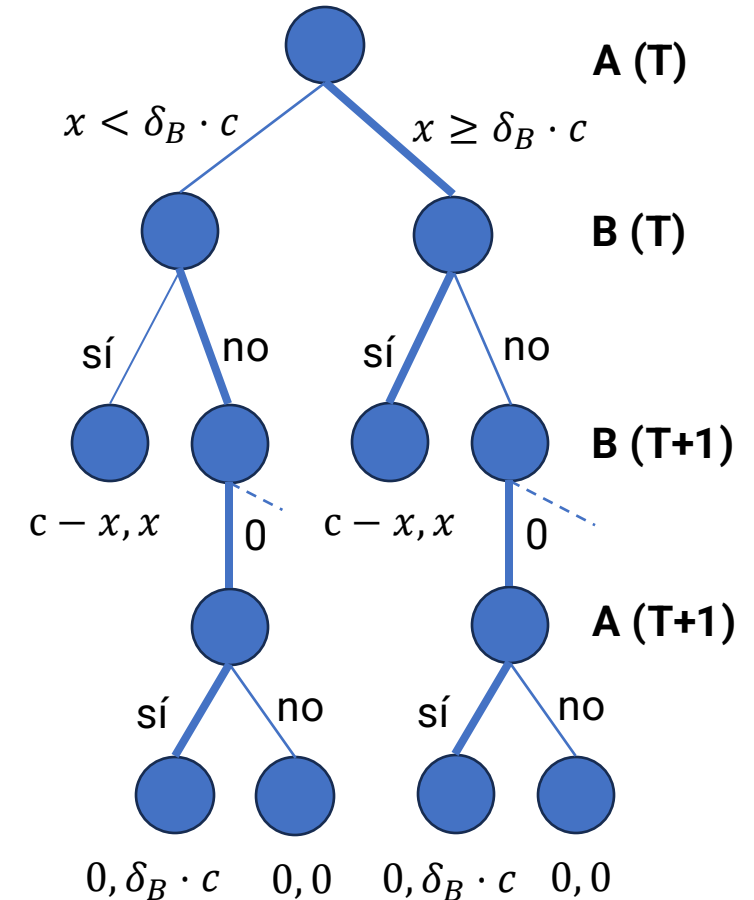
- B ofrece 0 (único equilibrio), A acepta: $(\delta_A \cdot 0, \delta_B \cdot (c - 0)) = (0, \delta_B \cdot c)$

Turno T (respuesta de B):

- Oferta que B acaba de realizar: x
- Si $x \geq \delta_B \cdot c$, B acepta (igual o supera el equilibrio en T)
- Si $x < \delta_B \cdot c$, B rechaza (empeora el equilibrio en T)

Turno T (oferta de A):

- A no puede ofrecer 0 porque en el último turno manda B con ventaja
- A ofrece la cantidad mínima que puede prevenir el rechazo de B : $\delta_B \cdot c$



Solución de regateo de Nash

- El algoritmo de inducción hacia atrás garantiza encontrar equilibrios pero **no hay garantías de óptimos de Pareto ni acuerdo *razonable***
- Una función de regateo de dos agentes F (que retorna recompensas en base a un problema de regateo: $F(S, d) = x$) es **razonable** si **cumple los siguientes axiomas de regateo de Nash**:
 - **Covarianza afín**: $F(\Gamma(S), \Gamma(d)) = \Gamma(F(S, d))$, siendo Γ una función de transformación lineal (no polinómica)
 - **Pareto-eficiencia**: $F(S, d) = x \wedge [\exists x' \in S: (\forall i: x'_i \geq x_i) \rightarrow x = x']$
 - **Simetría**: $[d_1 = d_2 \wedge ((x_1, x_2) \in S \rightarrow (x_2, x_1) \in S)] \rightarrow \exists a: F(S, d) = (a, a)$
 - **Independencia de alternativas irrelevantes**:
$$S \subseteq S' \wedge F(S', d) \in S \rightarrow [F(S, d) = F(S', d)]$$

Solución de regateo de Nash

- La **solución de regateo de Nash** $F^N(S, d)$ es el problema de maximización:

$$\max \prod_{i=1}^2 (x_i - d_i)$$

- La solución a este problema de maximización para cualquier combinación de S y d , dadas las condiciones $x_1 \geq d_1$, $x_2 \geq d_2$ y $(x_1, x_2) \in S$, **cumple con todos los axiomas de regateo de Nash**

Juegos adversariales

Competición

Cuando no se busca un acuerdo

- La negociación depende de la existencia potencial de acuerdos
- **En juegos de intereses puramente conflictivos, y en especial en juegos de suma cero, no es posible presuponer la posibilidad de estos acuerdos**
- En general, la manera de resolver estas situaciones es mediante la resolución de juegos en forma extensiva, veremos:
 - Minimax
 - Expectiminimax
 - Árbol de búsqueda de Monte Carlo

Algoritmo Minimax

- Habíamos visto que el algoritmo de inducción hacia atrás tiene una complejidad muy alta que lo hace difícil de usar en general
- Sin embargo, si estamos en una situación puramente conflictiva, hay ciertas **optimizaciones** que podemos aplicar para reducir esta complejidad
- **Si no hay posibilidad de acuerdo, entonces podemos asumir que el comportamiento de cada agente será totalmente racional (desde el punto de vista individual)**
 - En un juego de suma cero, **maximizar nuestra utilidad es equivalente a minimizar la del adversario**
- Si asumimos que estamos en un juego de suma cero, podemos reducir el vector de recompensas a un solo valor: $(x_1, x_2) \rightarrow (x, -x)$
 - Podemos aplicar cotas inferiores y superiores a los nodos que exploramos para **podar**

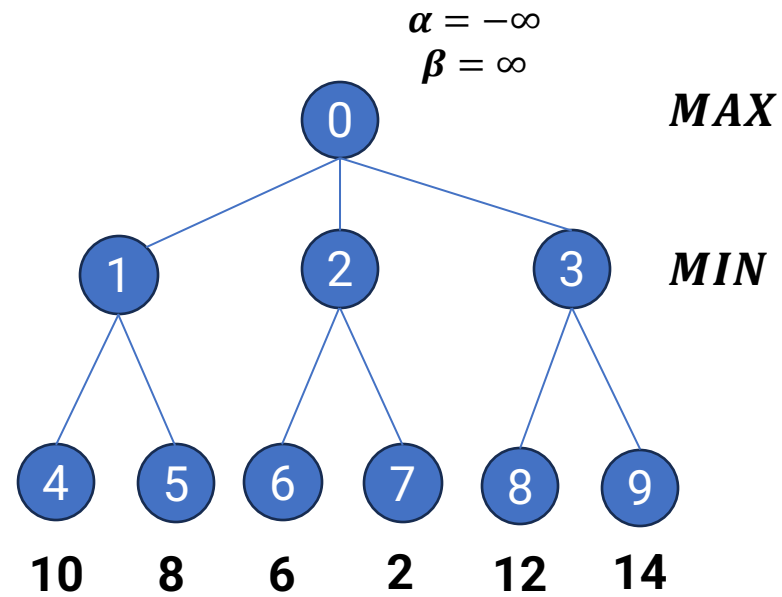
Algoritmo Minimax

- El **algoritmo Minimax** es una especialización de inducción hacia atrás: propagamos la (única) recompensa x desde los nodos terminales hacia la raíz
 - Cada decisión para el agente *MAX* busca maximizar x
 - Cada decisión para el agente *MIN* busca minimizar x (maximizar $-x$)
 - En cada nodo actualizamos y propagamos valores α y β que representan respectivamente la cota inferior y superior $\beta \geq x \geq \alpha$ sobre los equilibrios posibles
 - En cada nodo suyo, *MAX* dejará de explorar sucesores si el valor actual de *mejor_utilidad* es mayor o igual que la cota superior (β) porque *MIN* no los aceptará
 - En cada nodo suyo, *MIN* dejará de explorar sucesores si el valor actual de *mejor_utilidad* es menor o igual que la cota inferior (α) porque *MAX* no los aceptará

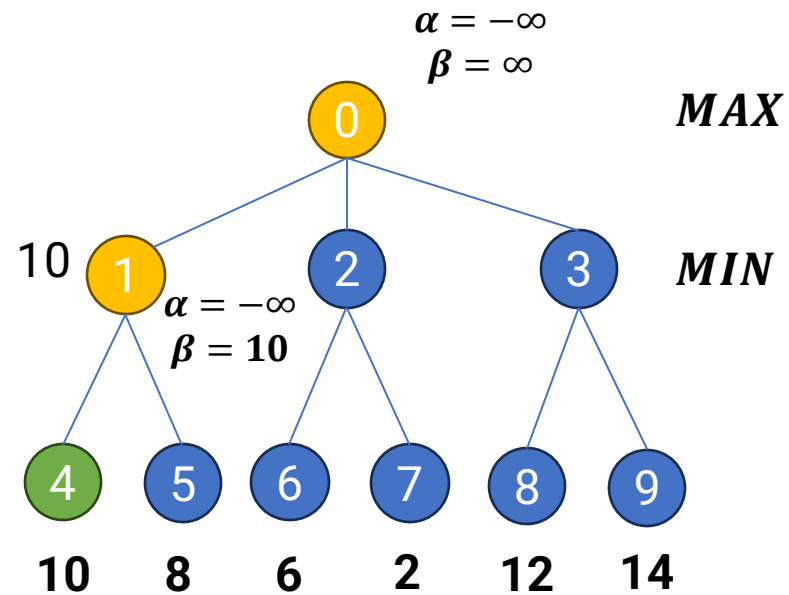
Algoritmo Minimax con poda alfa-beta

función Minimax(nodo h , real $\alpha = -\infty$, real $\beta = \infty$) **retorna** $u_A(h)$: recompensa para MAX
 si $h \in Z$ **entonces retorna** $u_{MAX}(h)$
 $mejor_utilidad \leftarrow (2\rho(h) - 3) \cdot \infty$ # $-\infty$ para MAX , ∞ para MIN
 para toda $a \in \mathcal{X}(h)$
 si $\rho(h) = MAX$ **entonces**
 $mejor_utilidad \leftarrow \max(mejor_utilidad, \text{Minimax}(\text{Sucesor}(h, a), \alpha, \beta))$
 si $mejor_utilidad \geq \beta$ **entonces retorna** $mejor_utilidad$
 $\alpha \leftarrow \max(\alpha, mejor_utilidad)$
 si no
 $mejor_utilidad \leftarrow \min(mejor_utilidad, \text{Minimax}(\text{Sucesor}(h, a), \alpha, \beta))$
 si $mejor_utilidad \leq \alpha$ **entonces retorna** $mejor_utilidad$
 $\beta \leftarrow \min(\beta, mejor_utilidad)$
 retorna $mejor_utilidad$

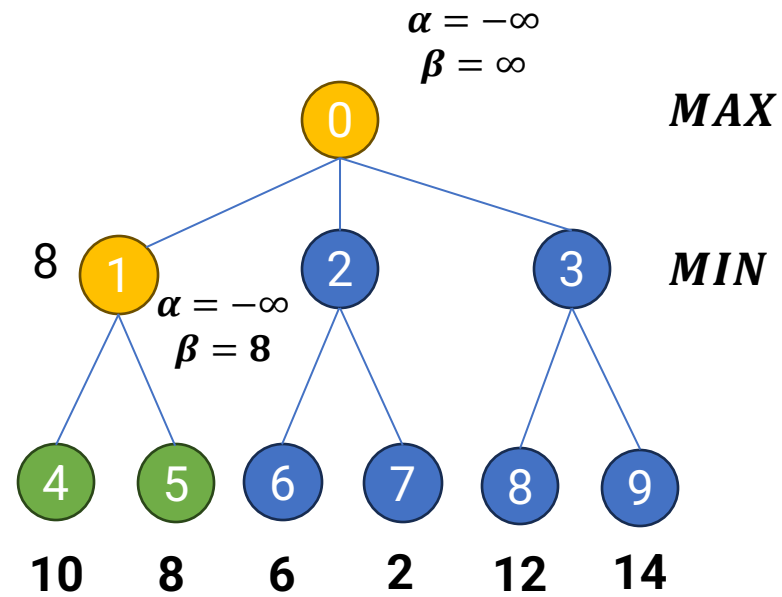
Algoritmo Minimax



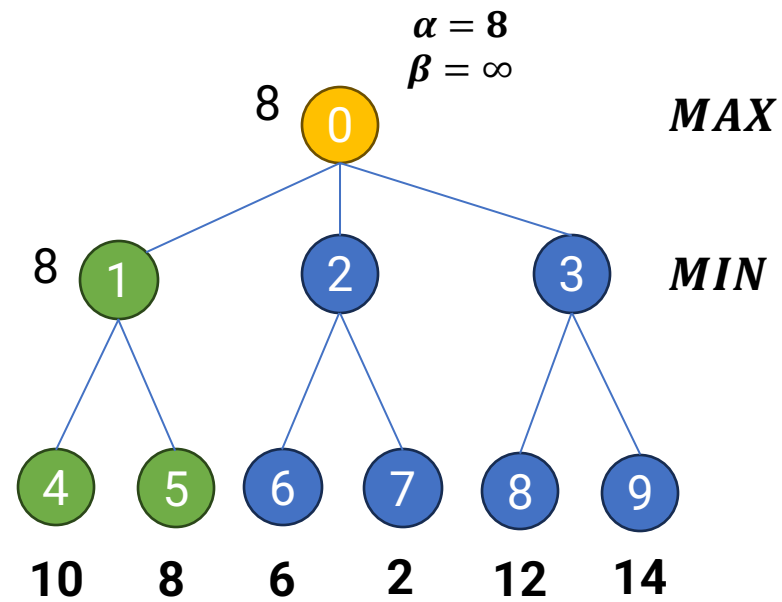
Algoritmo Minimax



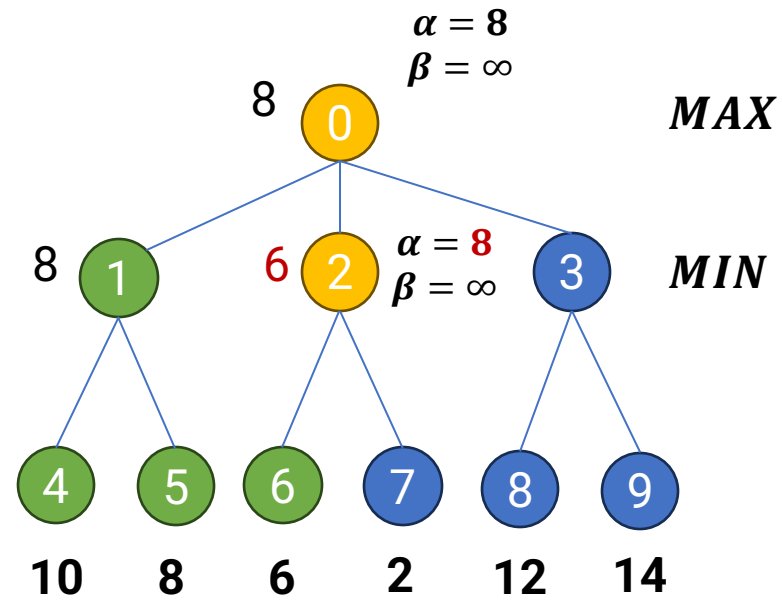
Algoritmo Minimax



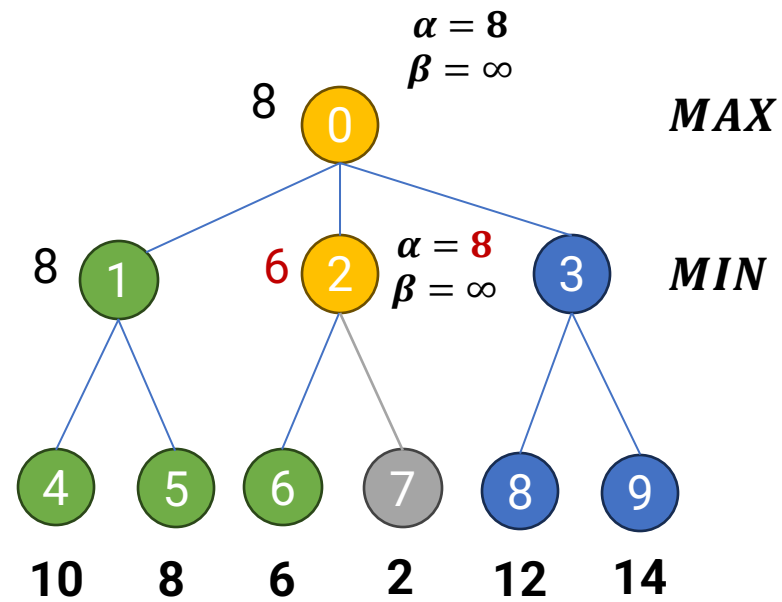
Algoritmo Minimax



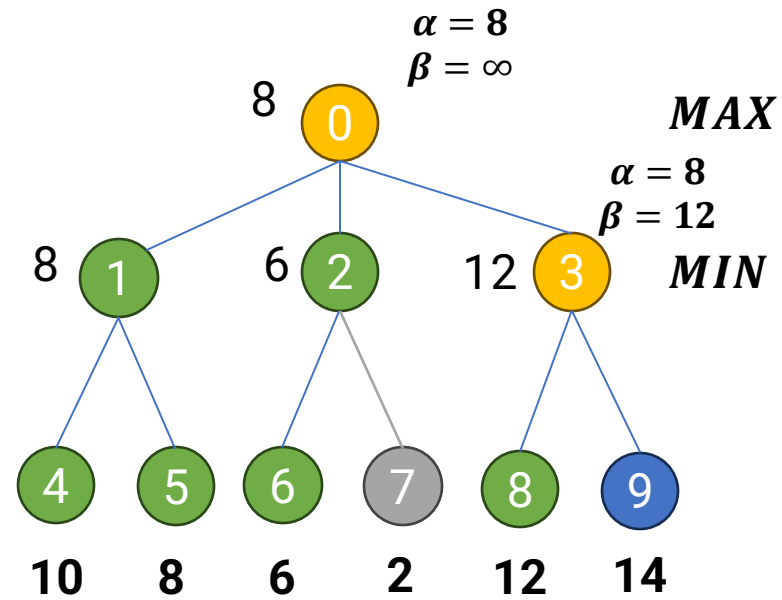
Algoritmo Minimax



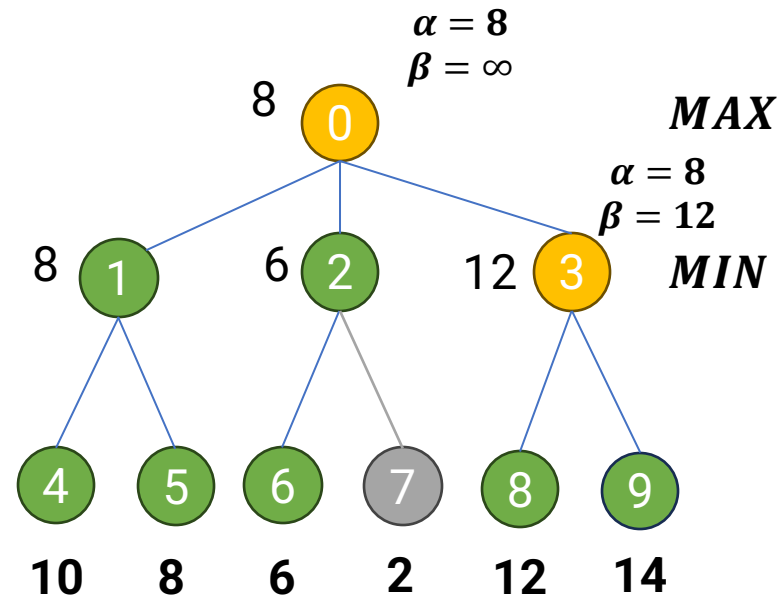
Algoritmo Minimax



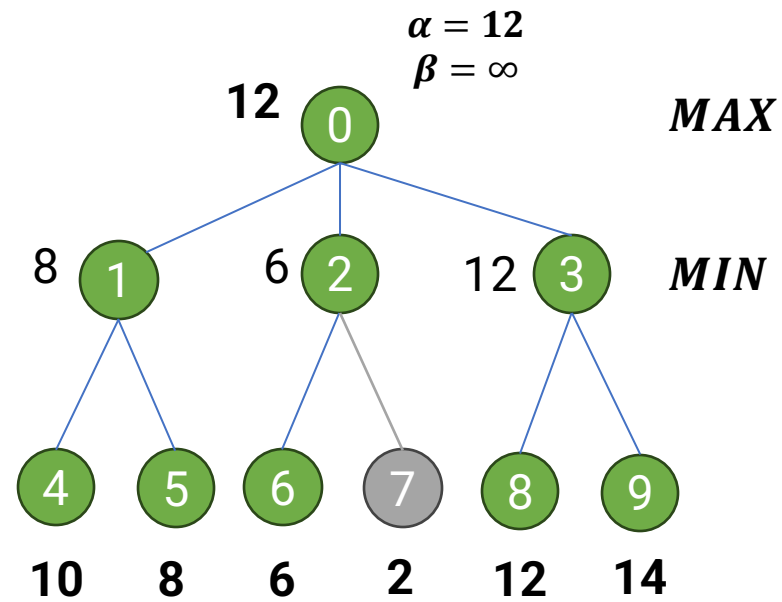
Algoritmo Minimax



Algoritmo Minimax



Algoritmo Minimax



Algoritmo Minimax

- El algoritmo Minimax es ampliamente utilizado en juegos de información perfecta (observabilidad total) entre dos jugadores
 - Juegos competitivos: ajedrez, Go, ...
 - Investigación operativa: economía, política, mercados financieros, deportes
 - Redes generativas adversariales (generador vs discriminador, *GANs*)
- La complejidad temporal en caso peor sigue siendo $O(b^m)$
 - Aunque, en el caso medio, **puede mejorar mucho** (hasta $O\left(b^{\frac{m}{2}}\right)$) **si podemos ordenar los nodos no terminales a visitar con conocimiento experto del entorno** (heurísticos)
- La complejidad espacial sigue siendo $O(b \cdot m)$

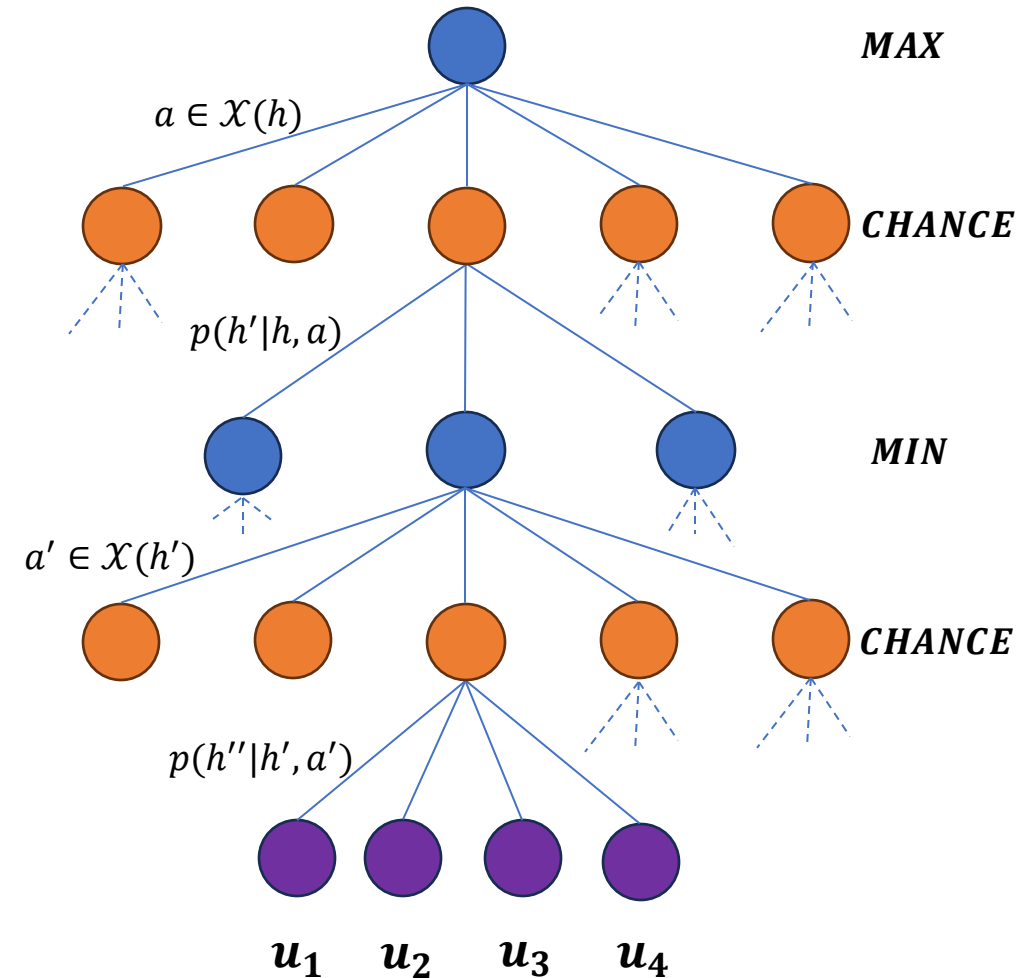
Algoritmo Minimax

- La dificultad en su aplicación reside principalmente en:
 - Es necesaria la suposición de observabilidad total
 - El entorno ha de ser determinista y no estocástico
 - No es directamente aplicable a juegos que no sean de suma cero
 - Escala muy mal en situaciones con un factor de ramificación alto
 - La complejidad o imposibilidad de valorar correctamente las utilidades de los otros jugadores en nodos no terminales

Expectiminimax

- Si el entorno es estocástico o parcialmente observable, es posible utilizar una variante de Minimax llamada Expectiminimax
 - El árbol de búsqueda incluye nodos estado-acción (*chance nodes*), de manera análoga a como se definen los MDPs
- La función de valor Expectiminimax, em , sobre un nodo h se define como:

$$em(h) = \begin{cases} u_{MAX}(h) & \text{si } h \in Z \\ \max_{a \in \mathcal{X}(h)} em(\text{Sucesor}(h, a)) & \text{si } \rho(h) = MAX \\ \min_{a \in \mathcal{X}(h)} em(\text{Sucesor}(h, a)) & \text{si } \rho(h) = MIN \\ \sum_{h'} p(h'|h, a) \cdot em(h') & \text{si } \rho(h) = CHANCE \end{cases}$$



Monte Carlo Tree Search

Competición

Árbol de búsqueda de Monte Carlo

- Una alternativa que solventa algunos de los inconvenientes de la inducción hacia atrás es el árbol de búsqueda de Monte Carlo
- Su aplicación es especialmente adecuada cuando:
 - Podemos realizar simulaciones aleatorias sobre el comportamiento del agente y el entorno
 - El espacio de búsqueda es intratable
 - El entorno es dinámico o estocástico
 - La dinámica del entorno es desconocida o demasiado compleja, y hacer una estimación heurística fiable es difícil
 - El tiempo de computación es limitado (racionalidad limitada por tiempo)

Fundamentos teóricos

- Ley de los Grandes Números:
 - en ensayos repetidos e independientes
 - con una probabilidad p de que suceda un evento $X = x$ en cada ensayo
 - **la diferencia entre**
 - la frecuencia con la que ocurre $X = x$, y
 - la probabilidad p
 - **converge a 0**, a medida que el número de ensayos tiende a infinito
- **Cuanto más ensayos hagamos, más podremos aproximarnos a:**
 - La esperanza del valor de la variable aleatoria X
 - Una varianza cercana a cero

Simulación de Monte Carlo

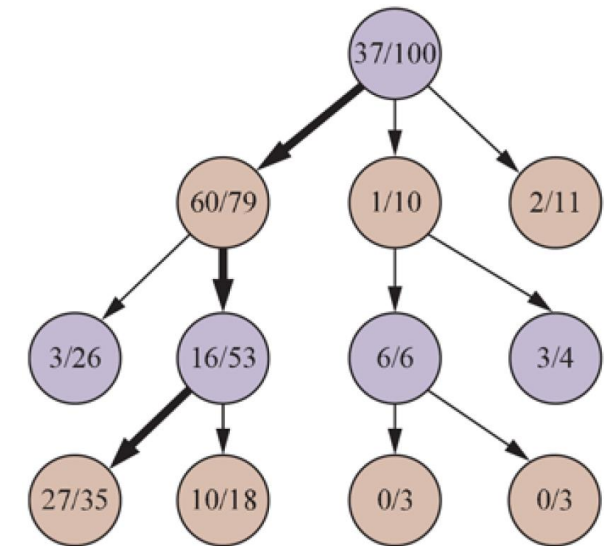
- Método para estimar el valor de una variable de manera estadística (von Neumann, Ulam)
 - Población: conjunto de posibles ejemplos
 - Muestra: subconjunto representativo de la población
 - Una muestra aleatoria tiende a exhibir las mismas propiedades que la población de la que se ha obtenido
- El método consiste en realizar un número potencialmente alto de ensayos independientes y aleatorios para aproximar dicho valor:
 1. Generación de muestras aleatorias (e.g. estados iniciales o semillas de experimentación)
 2. Simulación de un escenario particular a partir de las muestras aleatorias
 3. Cálculo de medias y varianzas (en base a alguna distribución de probabilidad)

Árbol de búsqueda de Monte Carlo

- Conceptos similares al método de simulación de Monte Carlo se aplican en problemas de búsqueda en el **Árbol de Búsqueda Monte Carlo (MCTS)**:
 - MCTS **aproxima la utilidad de ciertos nodos** (no necesariamente todos) del espacio de estados para obtener una estrategia/política
 - Esta aproximación estadística permite **capturar y modelar la dinámica del entorno y el comportamiento del oponente sin necesidad de heurísticos** (¡pero sí de capacidad de computación!)
 - El muestreo y los ensayos en MCTS no son totalmente aleatorios, ya que se parte en cada iteración de un árbol parcialmente construido que se va refinando progresivamente
 - En determinados puntos del algoritmo, sí se realiza **simulación**
- Cuatro fases: selección, expansión, simulación y propagación

Estructura del árbol

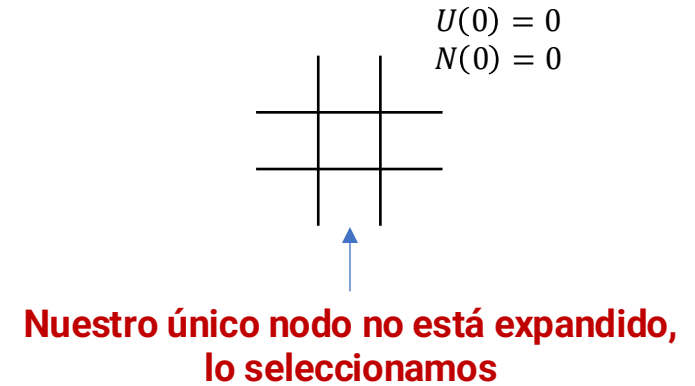
- Tenemos un árbol con nodos representando estados s y relacionados por las acciones a que cada agente puede tomar
 - Podemos acceder al nodo padre de cada nodo con la función $Parent(s)$
- A partir de un estado s y una acción a , podemos ir a un estado s' , si el entorno es determinista y no estocástico
 - Alternativamente, podemos modelar los nodos como pares estado-acción (*chance nodes*)
- Cada nodo estado s tiene asociados, como mínimo:
 - Las recompensas obtenidas por todas las iteraciones que han pasado por s , en una lista o como una acumulación $U(s)$ (suma)
 - El número de iteraciones que han pasado por s : $N(s)$
- Tenemos acceso a una función f que nos da el valor de cada nodo y que determina la mejor estrategia a partir de cada nodo s , por ejemplo la media aritmética: $f(s) = \frac{U(s)}{N(s)}$ si $N(s) > 0$, $f(s) = 0$ si $N(s) = 0$



Artificial intelligence: a modern approach (Russell & Norvig), Fig. 5.10

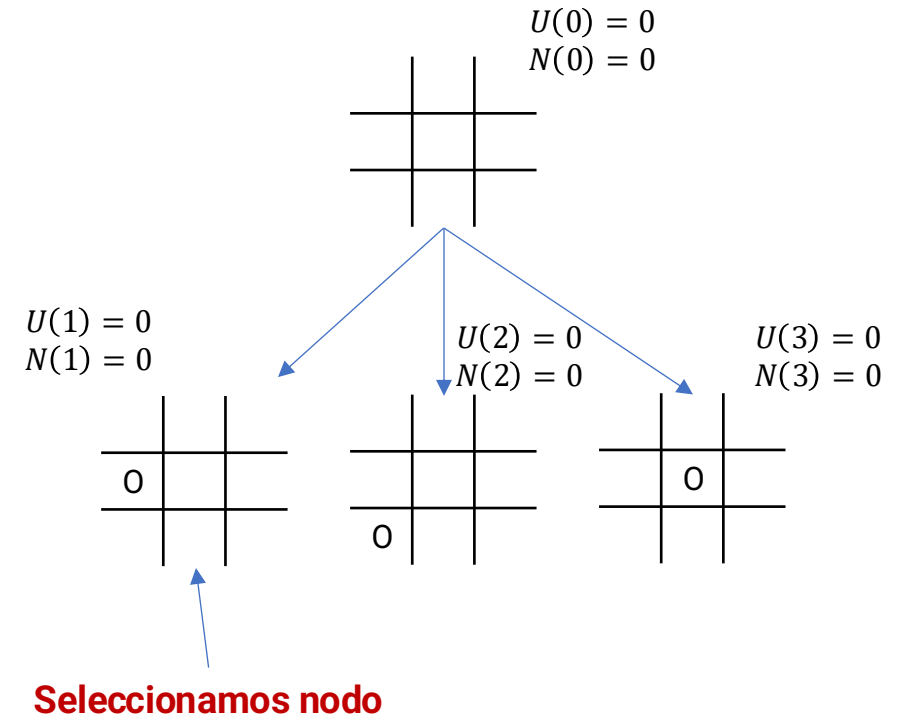
Fase 1: selección

- Recorremos el árbol en la dirección del mejor nodo (usando una función de exploración g) hasta encontrar un nodo que:
 - No esté expandido, o
 - No tenga hijos (e.g. sea terminal)
- Para la función de exploración g , podemos asumir de momento que $g = f$



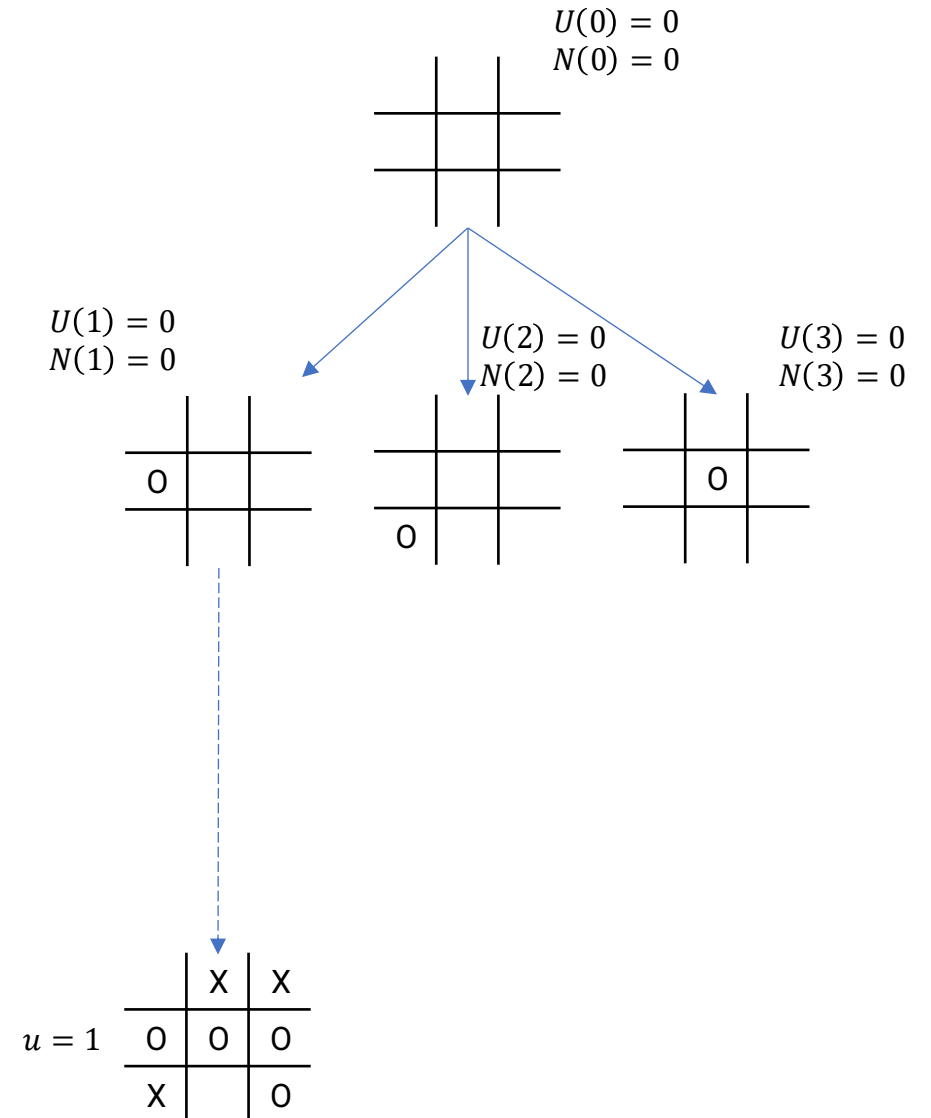
Fase 2: expansión

- Si el nodo seleccionado no es terminal, obtenemos un nuevo nodo sucesor para cada acción disponible en el nodo
- Los nodos pueden representar estados o pares estado-acción (en este ejemplo optamos por estados)
- Si los nodos representan estados, simulamos cada acción para obtener el nuevo estado para cada acción
- Si hemos expandido, seleccionamos el nuevo nodo aleatoriamente entre los nuevos sucesores



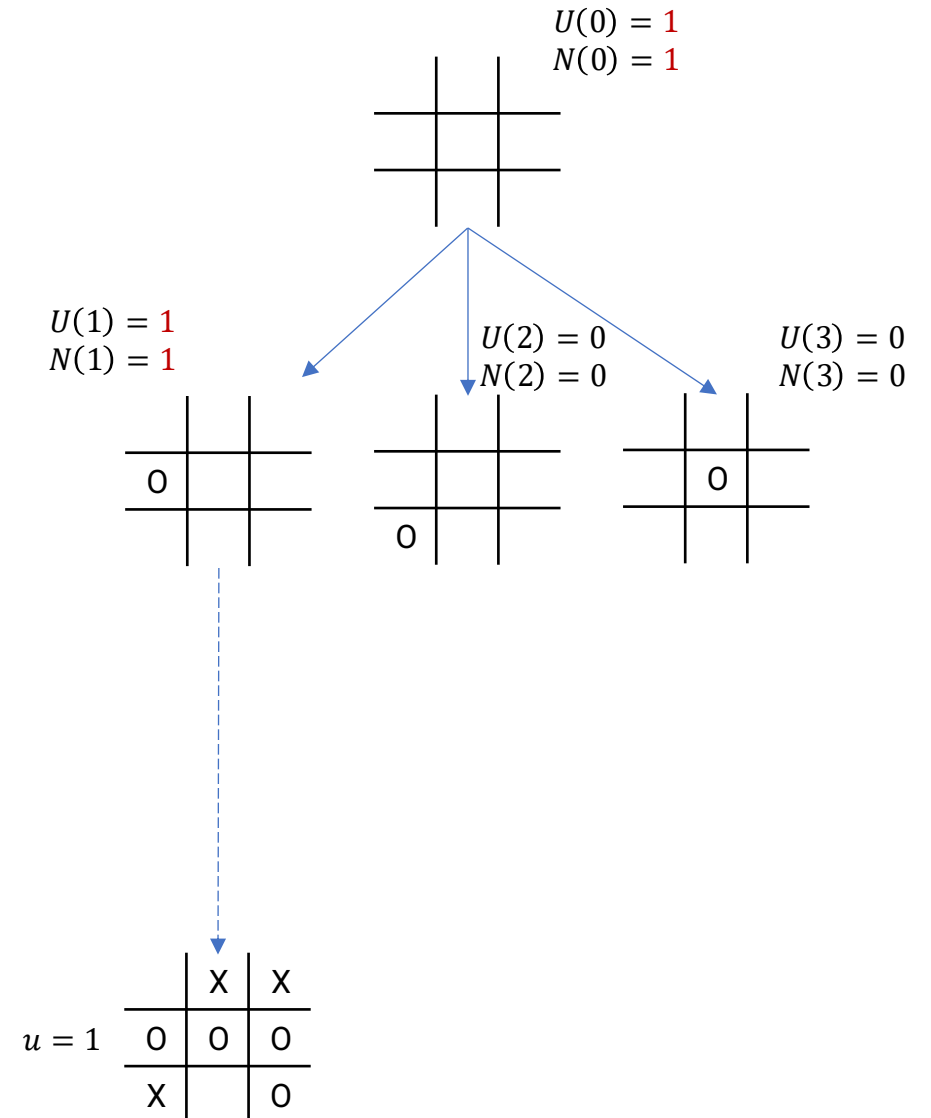
Fase 3: simulación

- Simulamos una ejecución completa a partir del nodo seleccionado, sin guardar el árbol de búsqueda
- Todas las acciones tomadas por todos los agentes en esta simulación son aleatorias



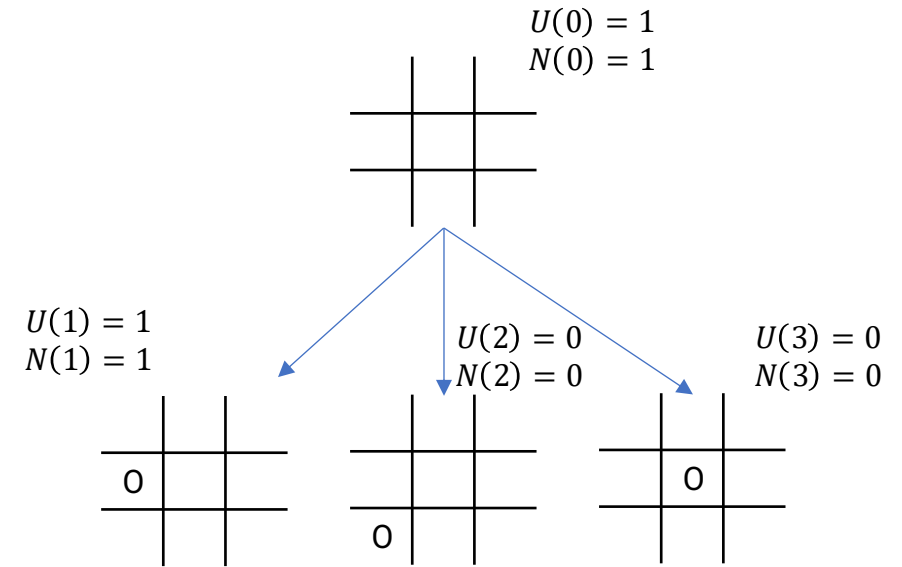
Fase 4: propagación

- Propagamos la utilidad del nodo terminal alcanzado por la simulación a través del nodo seleccionado y todos sus antecesores hasta llegar al nodo raíz (sin incluirlo, es irrelevante)
- Esta propagación modificará las utilidades acumuladas y los números de visitas



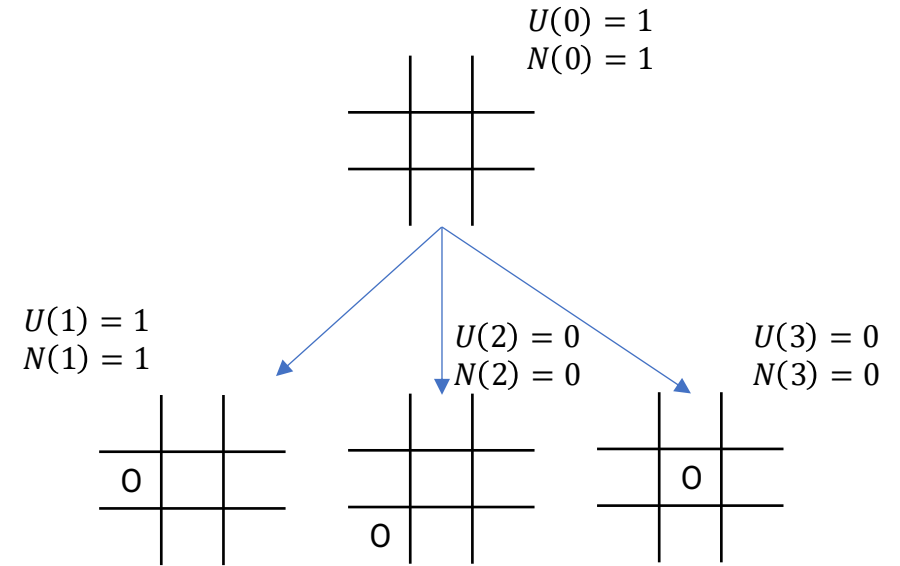
Fase 1: selección

- Recorremos el árbol en la dirección del mejor nodo (usando una función de exploración g) hasta encontrar un nodo que:
 - No esté expandido, o
 - No tenga hijos (e.g. sea terminal)
- ¿Tiene sentido que la función de exploración sea $g = f$?



Fase 1: selección

- Recorremos el árbol en la dirección del mejor nodo (usando una función de exploración g) hasta encontrar un nodo que:
 - No esté expandido, o
 - No tenga hijos (e.g. sea terminal)
- ¿Tiene sentido que la función de exploración sea $g = f$?
 - ¡No! Necesitamos un equilibrio entre exploración y explotación

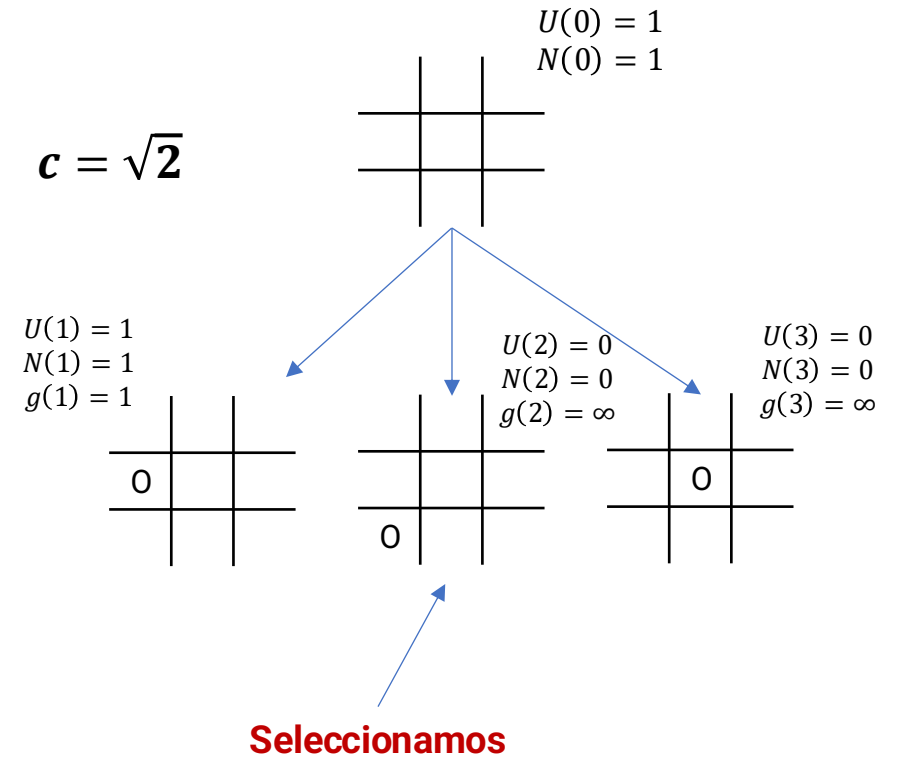


Fase 1: selección

- Recorremos el árbol en la dirección del mejor nodo (usando una función de exploración g) hasta encontrar un nodo que:
 - No esté expandido, o
 - No tenga hijos (e.g. sea terminal)
- Para la función de exploración g , podemos usar la función UCB1 (*Upper Confidence Bound for Trees*, c es el coeficiente de exploración):

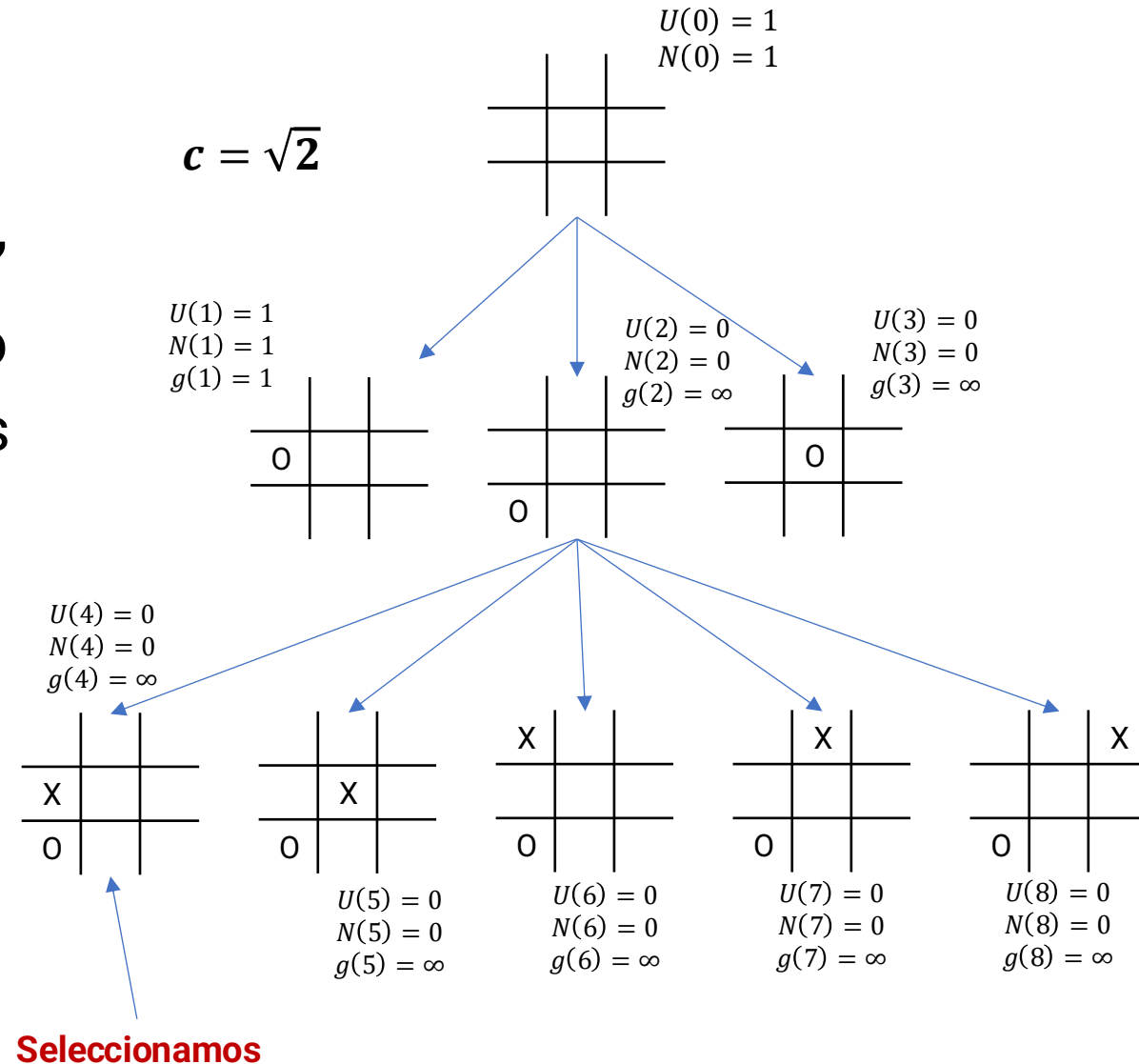
$$UCB1(s) = f(s) + c \cdot \sqrt{\frac{\log(N(\text{Parent}(s)))}{N(s)}}$$

- Esta función proviene de la literatura en *multi-armed bandits* y cumple ciertas garantías de convergencia y exploración



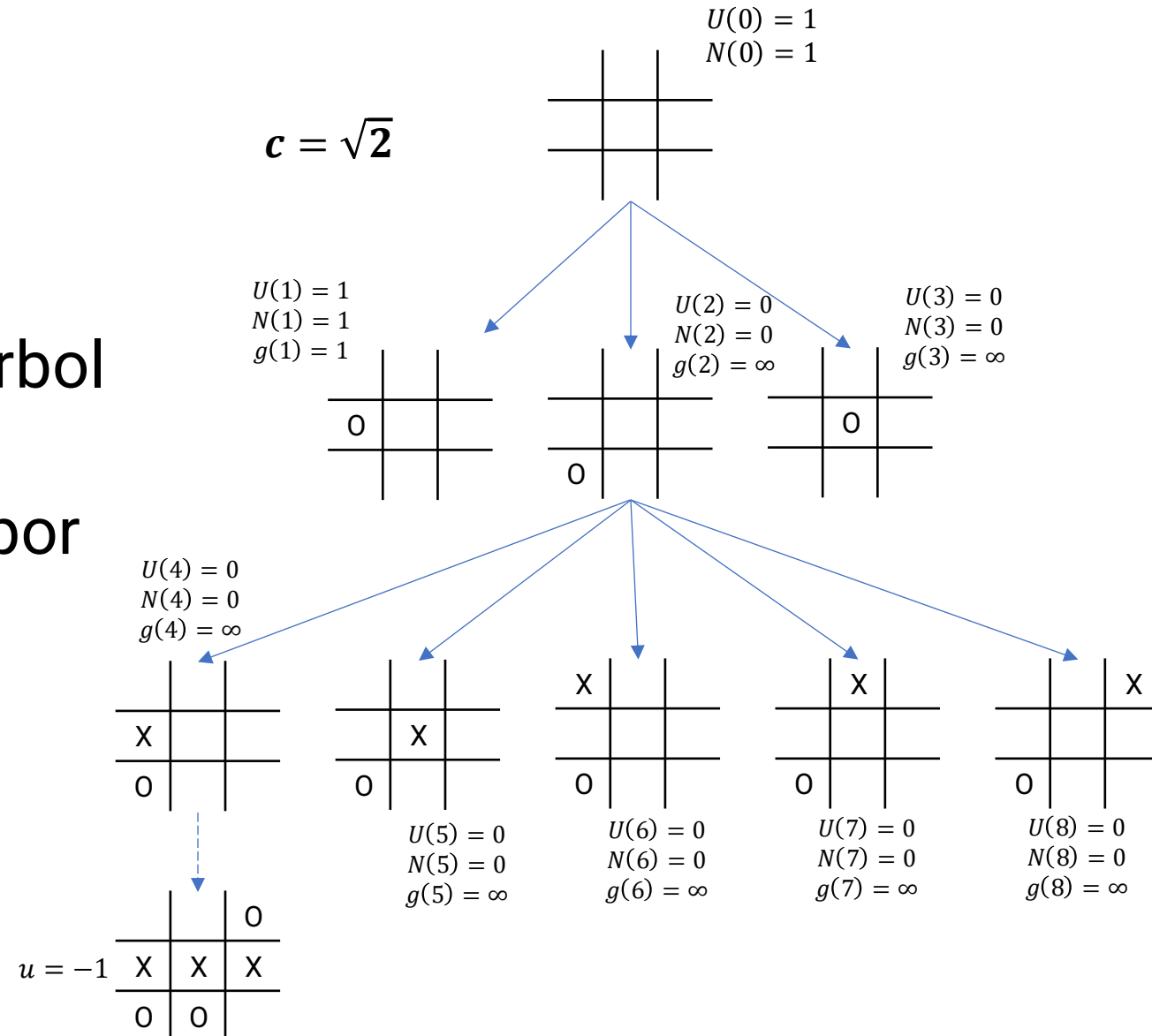
Fase 2: expansión

- Si el nodo seleccionado no es terminal, obtenemos un nuevo nodo sucesor para cada acción disponible en el nodo
- Los nodos pueden representar estados o pares estado-acción (en este ejemplo optamos por estados)
- Si los nodos representan estados, simulamos cada acción para obtener el nuevo estado para cada acción
- Si hemos expandido, seleccionamos el nuevo nodo aleatoriamente entre los nuevos sucesores



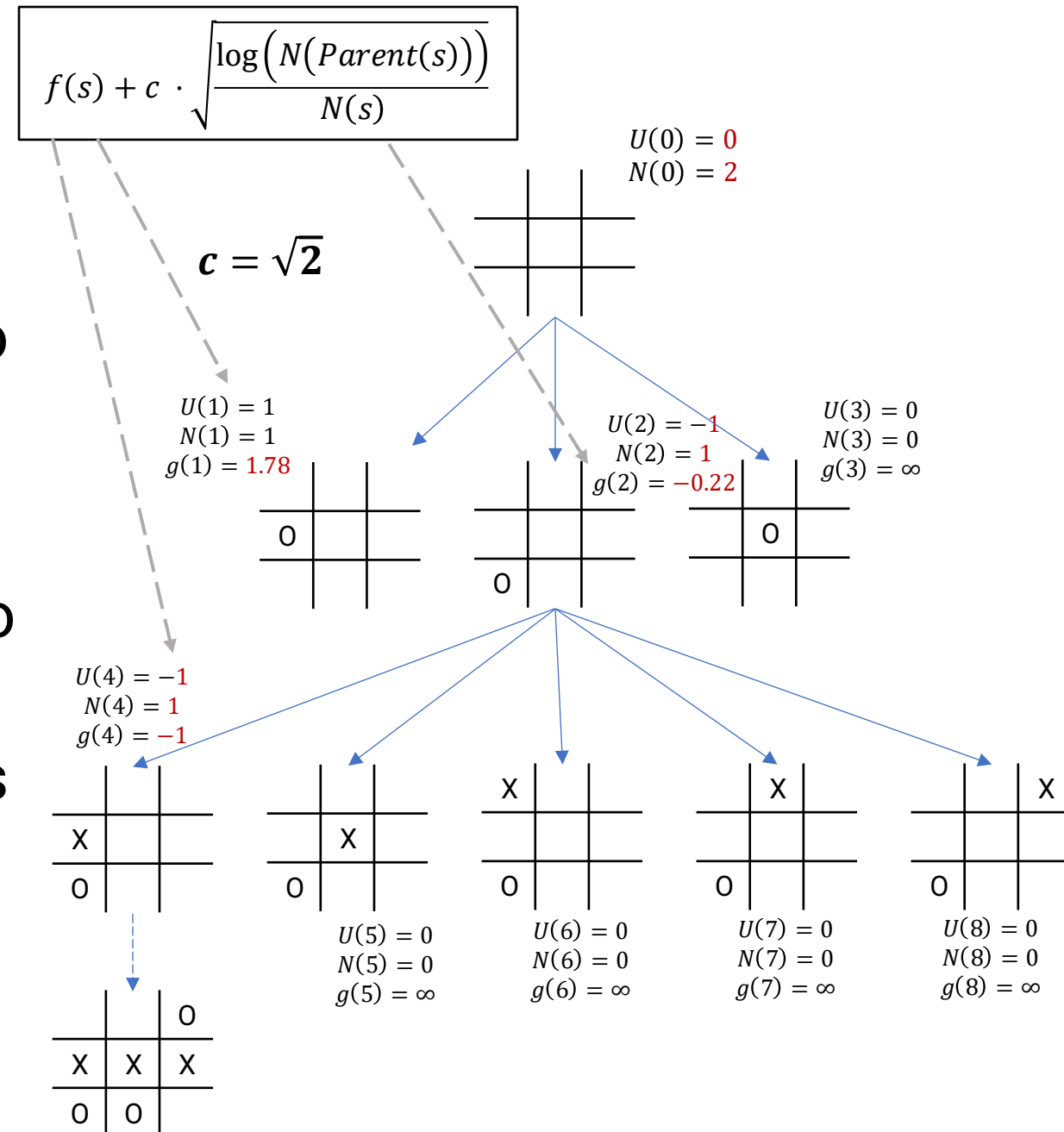
Fase 3: simulación

- Simulamos una ejecución completa a partir del nodo seleccionado, sin guardar el árbol de búsqueda
- Todas las acciones tomadas por todos los agentes en esta simulación son aleatorias



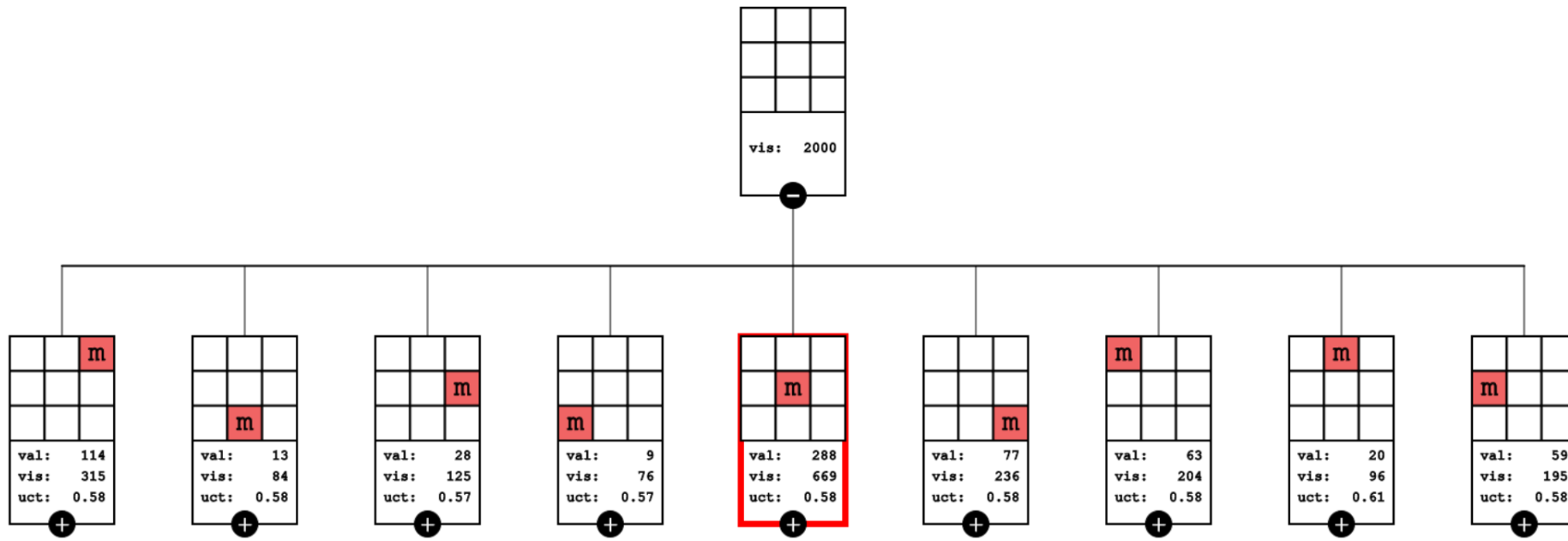
Fase 4: propagación

- Propagamos la utilidad del nodo terminal alcanzado por la simulación a través del nodo seleccionado y todos sus antecesores hasta llegar al nodo raíz (sin incluirlo, es irrelevante)
- Esta propagación modificará las utilidades acumuladas y los números de visitas



Simulación completa

- Podéis ver los resultados de la ejecución del algoritmo para este juego aquí: <https://vgarciasc.github.io/mcts-viz/>



Características de Monte Carlo

- La finalización puede ser por un **número máximo de iteraciones**, o también puede ser **por tiempo**
 - Esto permite el uso de este algoritmo para restricciones de tiempo disponible
- Es un algoritmo flexible
 - Los nodos pueden representar estados o **pares estado-acción**
 - Esto permite que se pueda modelar **comportamiento estocástico o desconocido**
 - Si es necesario, también se puede modelar sólo el comportamiento del agente que aprende, tratando los demás **adversarios como parte del entorno**
 - Hay variantes que **guían la simulación**, aplicando algún tipo de heurística en vez de aplicar acciones totalmente aleatorias
- El método de simulación de Monte Carlo o MCTS se pueden aplicar a diferentes tipos de algoritmo de aprendizaje por refuerzo, para guiar la exploración, e.g. como reemplazo de estimación directa, o aplicado a REINFORCE





AlphaZero shocked the chess world again with new results today.

AlphaZero Crushes Stockfish In New 1,000-Game Match



Pete  

Updated: Apr 17, 2019, 5:42 PM |  352 | Chess Event Coverage

 English 

In news reminiscent of the initial **AlphaZero** [shockwave](#) last [December](#), the artificial intelligence company **DeepMind** released astounding results from an updated version of the machine-learning chess project today.

The results leave no question—once again—that AlphaZero plays some of the strongest chess in the

<https://www.chess.com/news/view/updated-alphazero-crushes-stockfish-in-new-1-000-game-match>

**Minimax + heurísticas
diseñadas por expertos
VS
MCTS + policy network +
value network**