Transparencias-del-tema-7-Grafos...

PruebaAlien

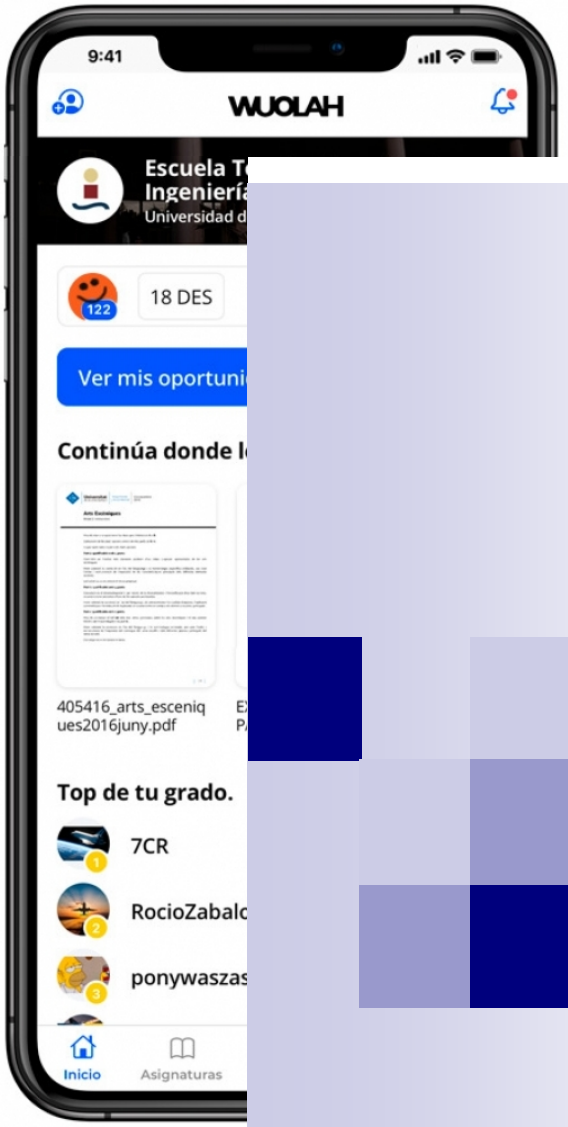
Algorítmica

2º Grado en Ingeniería Informática

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación
Universidad de Granada

KEEP CALM AND ESTUDIA UN POQUITO

WUOLAH + #QuédateEnCasa



Algorítmica

Capítulo 5: Algoritmos para la Exploración de Grafos.

Tema 13: Grafos implícitos

- Grafos Implícitos.
- Árboles para Juegos.
- Algoritmos de solución para juegos elementales

Introducción

Domingo 6 de octubre de 2019

32 EL PAÍS

SOCIEDAD

Kaspárov: "Fui el primer trabajador intelectual derrotado por una máquina"

'La Repubblica', EL PAÍS y otros diarios europeos debaten "un futuro ya presente"

JORDI PÉREZ COLOMÉ. Milán

Al ex campeón mundial de ajedrez Gari Kaspárov aún le escuece: "No fue la primera partida que perdí contra una máquina. Fue la primera partida que perdí", dijo. En 1997 *Deep Blue*, de IBM, ganó al campeón de ajedrez, que llevaba 12 años imbatido. "Las primeras semanas fueron muy duras", dice. "Fui el primer trabajador intelectual derrotado dolorosamente por una máquina delante de todo el mundo". Kaspárov admite que perdió no por la brillantez de *Deep Blue*, sino por su consistencia. Cometió menos errores. Aquella distancia no ha hecho más que crecer: "La distancia entre el campeón actual de ajedrez, Magnus Carlsen, y las máquinas es mayor que entre Usain Bolt y un Ferrari".

Kaspárov se ha convertido hoy en un evangelizador del futuro de la inteligencia artificial y de la colaboración entre hombre y máquina. Esta lección sobre la bondad del futuro se dio en la primera edición de Onlife, un encuentro organizado en Milán este viernes y sábado por el diario *La Repubblica* con la colaboración de Lena, la asociación de periódicos europeos a la que pertenece EL PAÍS. "Onlife" es un neologismo inventado por el filósofo italiano Luciano Floridi, que significa ese espacio donde "no hay una diferencia real entre estar *online* y *offline*, y que es una gran zona híbrida, rebautizada como *onlife*", según Carlo Verdelli, director de *La Repubblica*.

Durante el día y medio que duró Onlife pasaron más de 4.000 personas por el Teatro Parenti y el Politécnico de Milán para asistir a debates sobre un futuro que cada vez más ya está aquí: "Es un futuro ya presente", según Verdelli. Las charlas analizaron los retos de una época en la que la promesa de una tecnología liberadora se ha convertido en una realidad ambigua. Junto a Kaspárov, hablaron la alcaldesa de Barcelona, Ada Colau, y el alcalde de Milán, Giuseppe Sala; el creador de la app Waze que ayuda a evitar atascos, Uri Levine; los escritores Alessandro Baricco y Roberto Saviano; el matemático que mandó el primer paquete de datos por Internet, Leonard Kleinrock, y la autora del libro germinal *La era del capitalismo de la vigilancia*, Soshana Zuboff, entre otros.

Kaspárov, durante el foro Onlife, en Milán. / ALBERTO CATTANEO (FOTOGRAMMA)

"La tecnología es la razón principal por la que estamos vivos para quejarnos de la tecnología", bromeó Kaspárov. Su optimismo le llevó a comparar el valor de la inteligencia artificial con el de Steve Jobs, fundador de Apple. "Las máquinas pueden personalizarte y darte lo que no sabes que quieres, pero no saben darte lo que aún no existe, como hizo Jobs".

Otros ponentes fueron más cautos. Uno de los pioneros de Internet, Kleinrock, admitió que la evolución del medio ha sido irregular. En los primeros años era difícil imaginar este declive hacia una red donde el spam, los ataques y la falta de privacidad tuvieran un peso tan grande: "En los primeros 20 años nadie monetizó la Red y nadie se portaba mal", explicó. Les fue más difícil prever el declive.

La alcaldesa Colau explicó la labor de Barcelona contra "la gentrificación bestial" provocada por la plataforma Airbnb. "La primera vez que hablamos con ellos nos dijeron que no podíamos hacer nada en contra de ellos, pero luego aprendimos", dijo.

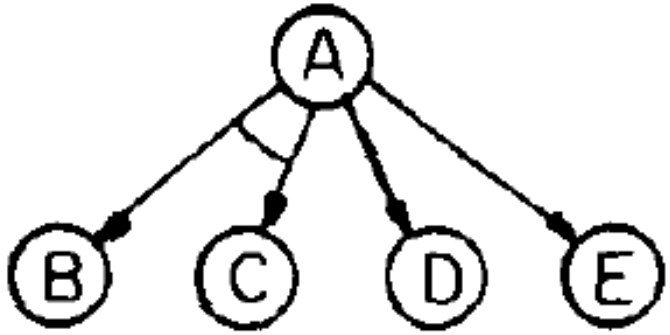
La mutación que vive la sociedad desde hace escasas tres décadas es varias cosas a la vez: una amenaza, una esperanza y un lío. Todas son válidas. Unos prefieren centrarse en lo que se pierde, otros en lo que se gana y otros aún en la complejidad.

Grafos implícitos. Grafos Y/O

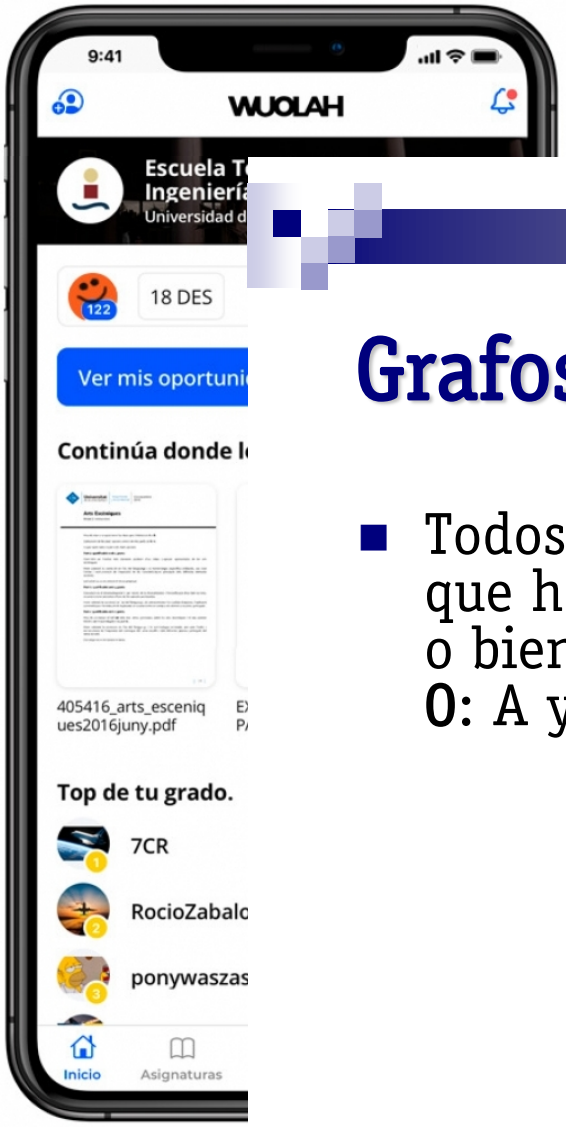
- Muchos problemas complejos pueden descomponerse en una serie de subproblemas tales que la solución de todos, o algunos de ellos, constituya la solución del problema original.
- A su vez, estos subproblemas pueden descomponerse en sub-subproblemas, y asi sucesivamente, hasta conseguir problemas lo suficientemente triviales como para poderlos resolver sin dificultad.
- Esta descomposición de un problema en subproblemas puede representarse mediante un grafo dirigido, en el que los nodos serían problemas, y los descendientes de un nodo serían los subproblemas asociados al mismo.
- Es un tema absolutamente actual cuyas aplicaciones son importantisimas: www, “data mining”, recuperación de información, ...

Grafos Y/O

- Por ejemplo, el grafo de la figura

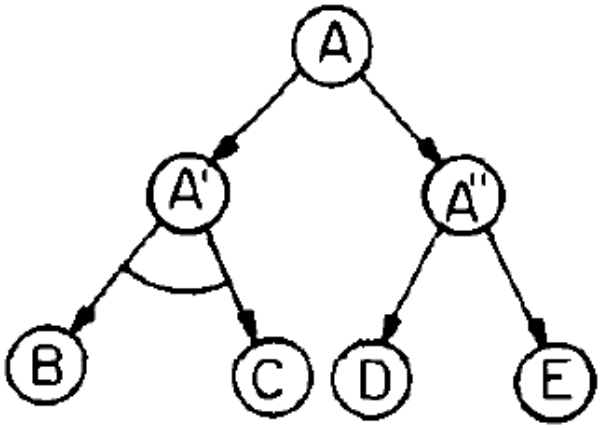


- representa un problema A que puede resolverse bien mediante la resolución de los dos subproblemas B y C, o bien mediante la de D o E aisladamente.
- Los grupos de subproblemas que deben resolverse para implicar una solución del nodo padre, se conectan mediante un arco que una las respectivas aristas (caso de B y C aquí).
- Podemos introducir nodos fantasma para uniformar el grafo



Grafos Y/O

- Todos los nodos serán tales que sus soluciones requieran que haya que resolver todos sus descendientes (**nodos Y**: A'), o bien que solo haya que resolver un descendiente (**nodos O**: A y A").

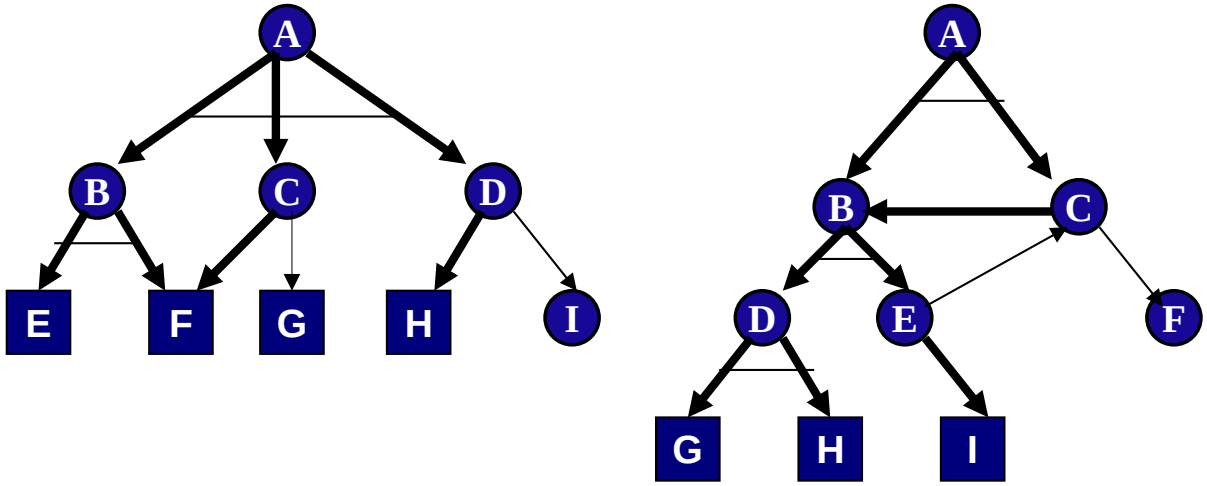


- Los nodos sin descendientes se llamarán terminales.
- Los nodos terminales representan problemas primitivos, y se marcan como resolubles o no resolubles. Los nodos terminales resolubles se representaran por rectángulos.

Grafos Y/O

- La descomposición de un problema en diversos subproblemas se conoce con el nombre de Reducción del Problema.
- La Reducción de Problemas se usa frecuentemente en ámbitos como la demostración de teoremas, la integración simbólica o el análisis de calendarios industriales.
- Cuando se usa la Reducción de Problemas, dos problemas diferentes pueden generar un subproblema común.
- En ese caso puede ser deseable tener solo un nodo representando a este subproblema, ya que esto podría significar tener que resolver ese subproblema solo una vez
- Lo vemos en la siguiente transparencia

Grafos Y/O



Nótese que el grafo no es un árbol.

Puede ser además que tales grafos tengan ciclos dirigidos, como en la figura derecha. La presencia de un ciclo dirigido no implica en si misma la irresolubilidad del problema.

Llamaremos grafo solución a un subgrafo de nodos resolubles que muestra que el problema esta resuelto.

Los grafos solución de los grafos de la anterior figura se representan por aristas gruesas.

Determinando la resolubilidad

- Veamos primero como determinar si un árbol Y/O dado representa o no un problema resoluble (la extensión al caso de grafos se deja como ejercicio).
- Está claro que, realizando un recorrido en post-orden del árbol Y/O, podemos determinar si un problema es resoluble o no.
- En lugar de evaluar todos los hijos de un nodo, el algoritmo finalizaría tan pronto como descubriera que un nodo es irresoluble o resoluble.
- Esto reduce la cantidad de trabajo que el algoritmo tiene que hacer, sin que por ello la salida se afecte

Determinando la resolubilidad

```
Procedimiento RESUELVE (T)
    {T es un árbol Y/O con raíz T. T ≠ 0. El algoritmo devuelve 1 si  el
    problema es resoluble, y 0 en otro caso}
Begin
   Caso  :T es un nodo terminal:   If T es resoluble
                                   Then Return (1)
                                   Else Return (0)
         : T es un nodo Y:         For cada hijo S de T do
                                 If RESUELVE (S) = 0
                                 Then Return (0)
                                       Return (1)
         : En otro caso:           For cada hijo S de T do
                                 If RESUELVE (S) = 1
                                 Then Return (1)
                                      Return (0)
End
```

Generación de grafos implícitos

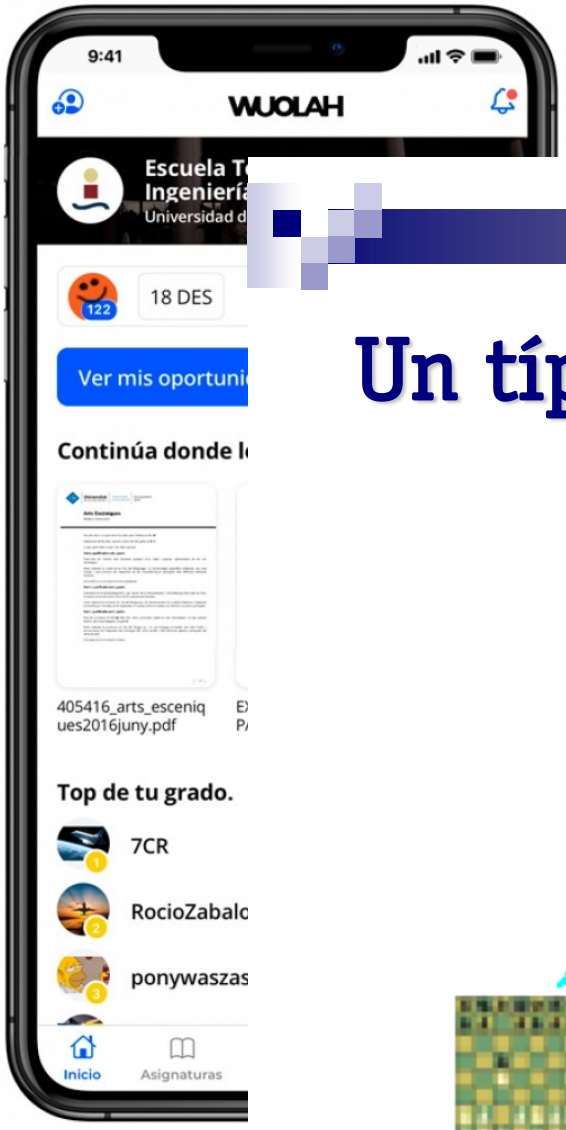
- A menudo, solo de dispone de forma implícita del árbol Y/O correspondiente a un problema.
- Por tanto, vamos a dar una función F que genere todos los hijos de un nodo ya generado.
- Los nodos del árbol pueden generarse primero en profundidad o primero en anchura.
- Como es posible que un árbol Y/O tenga profundidad infinita, si comenzáramos una generación del árbol primero en profundidad, generando todos los nodos en un camino infinito desde la raíz, podríamos no llegar a determinar un subarbol solución (incluso aunque existiera).
- Esto se evita restringiendo la búsqueda primero en profundidad a la generación del árbol Y/O dentro solo de una profundidad d.

Generación de grafos implícitos

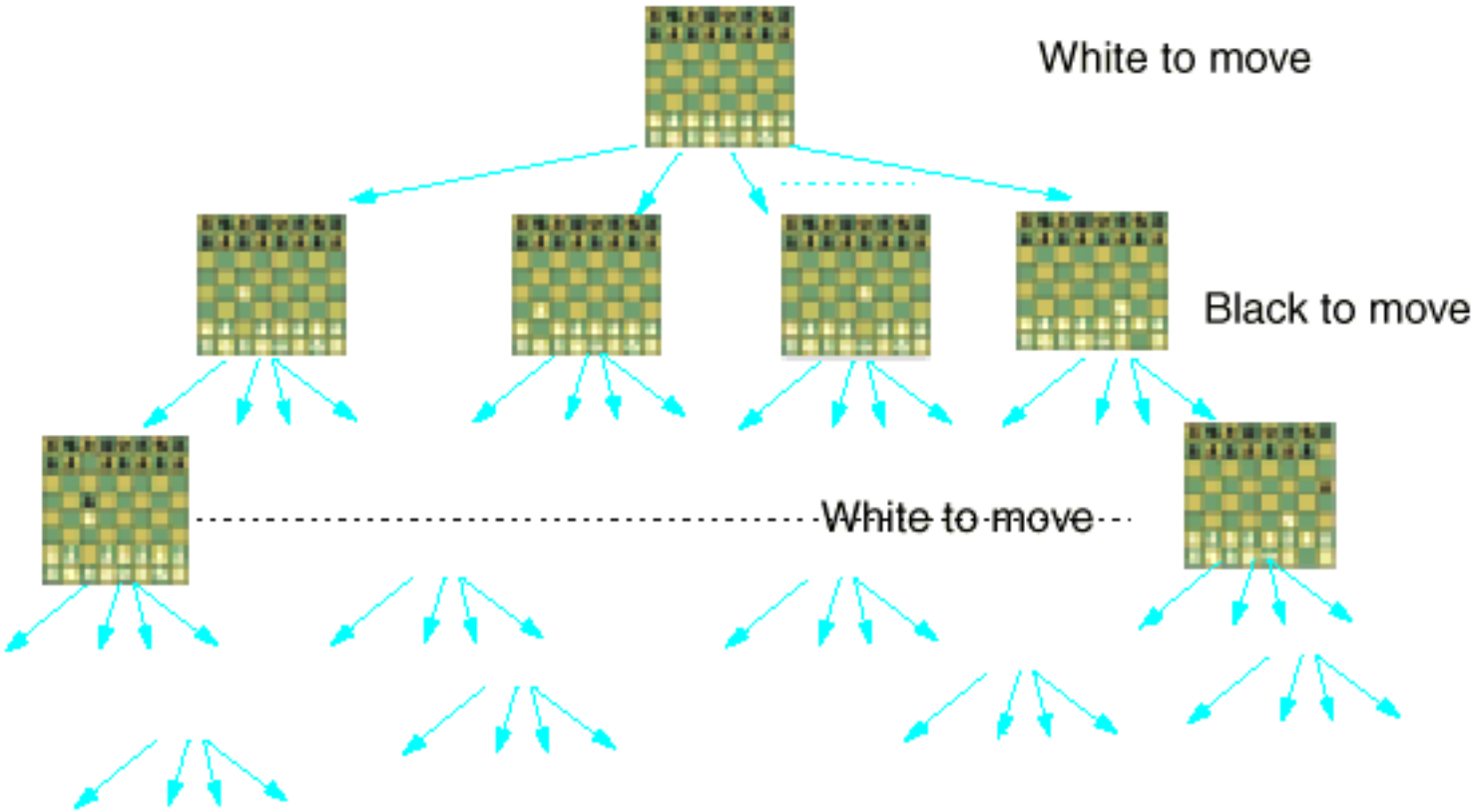
- Así, los nodos que a profundidad d sean no terminales, se etiquetarán irresolubles. En este caso, queda garantizado que la búsqueda primero en profundidad encontrará un subárbol solución, dado que hay una profundidad no mayor que d.
- La búsqueda (o generación) primero en anchura no presenta este inconveniente. Ya que cada nodo puede tener solo un número finito de hijos, ningún nivel en el árbol Y/O puede tener un número infinito de nodos.
- La generación primero en anchura del árbol Y/O garantiza encontrar un subárbol solución, si es que este existe.
- El algoritmo se deja como ejercicio

Árboles para juegos.

- La exploración de arboles se aplica continuamente en diferentes campos
- Una aplicación interesante de los árboles se encuentra en el desarrollo de juegos de estrategia (cartas, ajedrez, damas, etc.).
- Aunque solo sea desde el punto de vista de sus múltiples aplicaciones, el concepto de juego es uno de los mas interesantes que podemos encontrar.



Un típico árbol de juego



Un ejemplo clásico

- ¿Que movimiento debemos hacer?

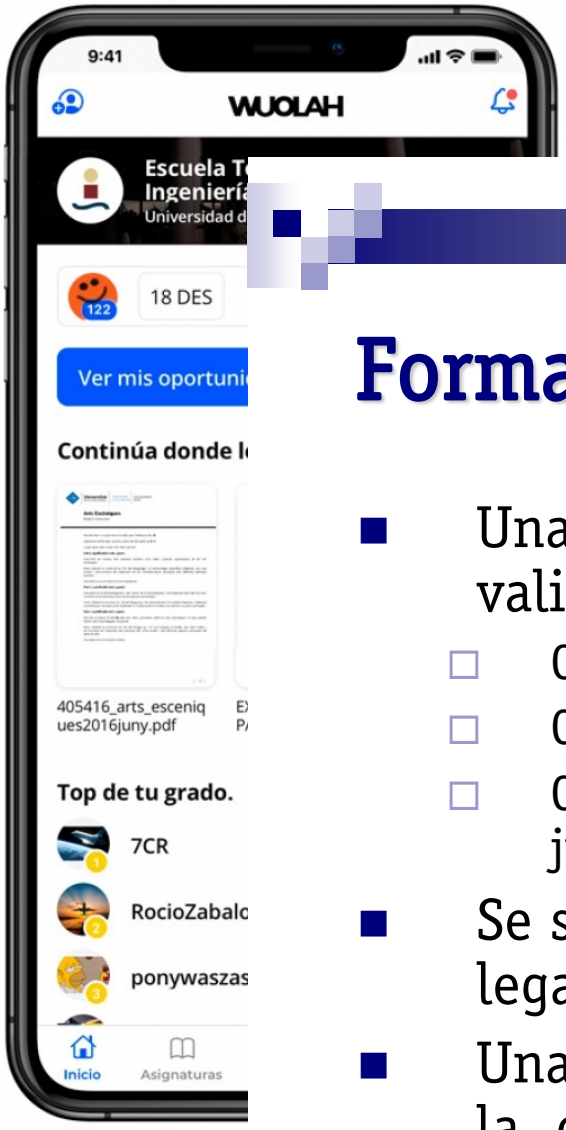
Mueven blancas

Ejemplo: juego de los palillos (Nim)

- Juegan dos jugadores A y B. El juego queda descrito por un panel que inicialmente contiene n palillos.
- Los jugadores A y B hacen alternativamente sus movimientos, empezando a jugar A. Un movimiento legal consiste en eliminar 1, 2 o 3 palillos del panel. Sin embargo, un jugador no puede quitar mas palillos de los que hay en el panel.
- El jugador que quita el último palillo pierde el juego y el otro gana.
- El panel esta completamente especificado en cualquier instante por el número de palillos que quedan, y en cualquier momento el status del juego se determina por la configuración del panel, junto con el jugador que le toca jugar a continuación.
- El juego de Nim no puede terminar en empate.

Ejemplo: juego de los palillos (Nim)

- Una configuración terminal es aquella que representa una situación de ganar, de perder o de empate. Todas las demás configuraciones son no terminales.
- En el juego de Nim solo hay una configuración terminal: Cuando no quedan palillos en el panel.
- Esta configuración es de ganancia para A, si B hizo el último movimiento o viceversa.
- Este es un juego para el que existe una forma óptima de jugar: Nunca se pierde
- El juego de Nim es importante porque es un juego de **Información Perfecta**, porque es un juego equitativo y porque cualquier otro juego equitativo es equivalente a uno de Nim.



Formalización: juegos mas generales

- Una sucesión C(1),...,C(m) de configuraciones se llama valida si
  - C(1) es la configuración inicial del juego
  - C(i), 0<i<m, son configuraciones no terminales, y
  - C(i+1) se obtiene de C(i) mediante un movimiento legal hecho por el jugador A si i es impar, y por el jugador B si i es par.
- Se supone que hay un número finito de movimientos legales.
- Una sucesión válida C(1),...,C(m) de configuraciones, en la que C(m) es una configuración terminal, se llama un caso del juego. La longitud de la sucesión C(1),...,C(m) es m.
- Un **Juego Finito** es un juego para el que no existen sucesiones válidas de longitud infinita.

Desarrollo de un juego general

- Todos los posibles casos de un juego finito pueden representarse por el árbol del juego.
- En el árbol del juego, cada nodo representaría una configuración, y el nodo raíz sería la configuración inicial C(1).
- Las transiciones desde un nivel al siguiente se hacen según mueva A o B.
  - En el juego de Nim las transiciones desde un nivel impar representarían los movimientos de A, mientras que todas las demás transiciones serian los movimientos de B.
- Las configuraciones terminales se representan por nodos hoja y suelen etiquetarse con el nombre del jugador que gana cuando se alcanza esa configuración.

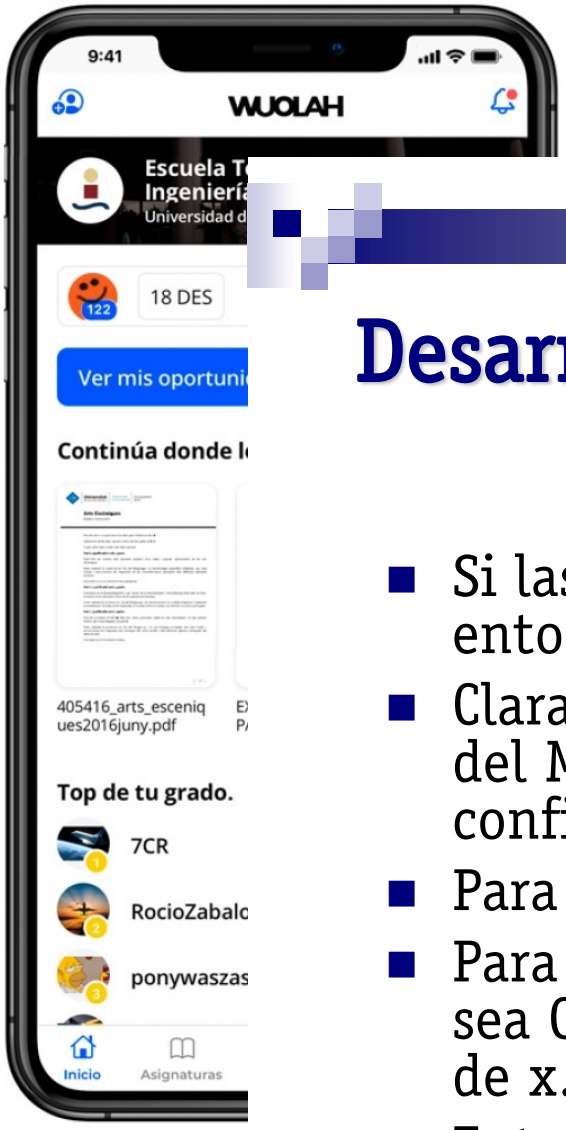
Desarrollo de un juego general

- El grado de cualquier nodo en el árbol del juego es igual, a lo mas, al número de movimientos legales distintos que haya.
- Por definición, el número de movimientos legales en cualquier configuración es finito.
- La profundidad de un árbol de juego es la longitud del mas largo caso del juego.
- Los árboles de juego son útiles para determinar el siguiente movimiento que un jugador debe hacer:
- Partiendo de la configuración inicial representada por la raíz, el jugador A se enfrenta con tener que elegir entre varios posibles movimientos.
- Supuesto que A quiera ganar el juego, A debería elegir el movimiento que maximice su posibilidad de ganar.

Desarrollo de un juego general

- Si la recompensa no está claramente definida, se puede usar una función de evaluación E(X) que asigne un valor numérico a cada configuración X.
- Esta función dará la recompensa para el jugador A asociada a la configuración X.
  - Para un juego como el Nim con pocos nodos, basta definir E(X) sobre las configuraciones terminales.
- E(x) = 1 si X es una configuración ganadora para A

  = -1 si X es una configuración perdedora para A
- Usando esta función de evaluación, ahora lo que queremos es determinar cual de las posibles configuraciones que el jugador A tiene como posibles (supongamos que sean b, c y d) debe elegir.



Desarrollo de un juego general

- Si las configuraciones a las que pede moverse son b, c y d entonces:
- Claramente, la elección será aquella cuyo valor sea el del Max{V(b), V(c), V(d)}, donde V(x) es el valor de la configuración x.
- Para nodos hoja x, V(x) se toma como E(x).
- Para todos los demás nodos x, sea $d \geq 1$ el grado de x, y sea C(1),...,C(d) la configuración representada por los hijos de x.
- Entonces V(x) se define como

$$V(x) = \text{Max } \{V[C(i)]\}\ 1 \leq i \leq d \text{ si x es un nodo de cuadrado (A)}$$
$$= \text{Min } \{V[C(i)]\}\ 1 \leq i \leq d \text{ si x es un nodo circular (B)}$$

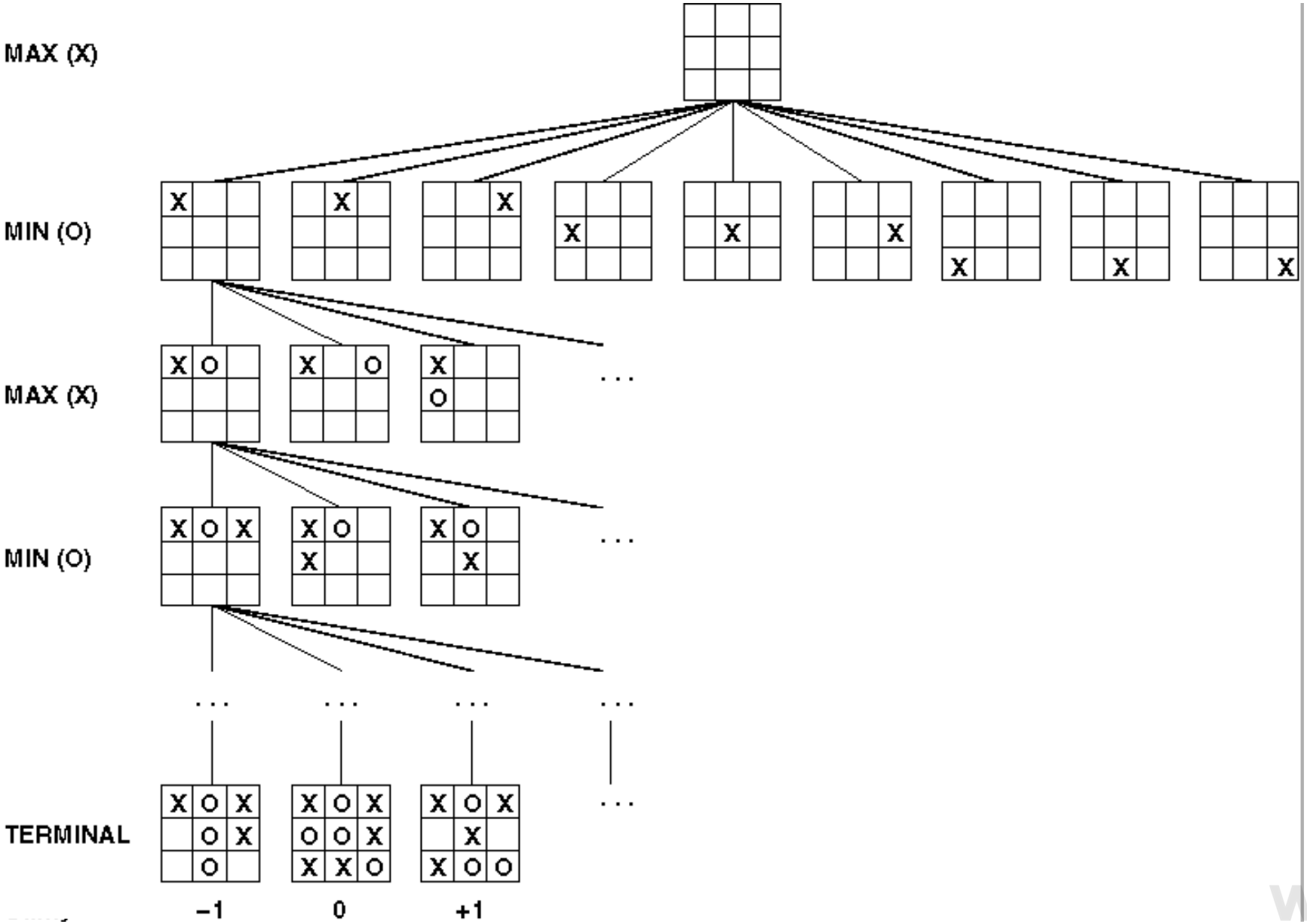
Generalización de la estrategia maximin

- La justificación de la formula anterior es simple.
- Si x es un nodo cuadrado, entonces estará en un nivel impar, y sería el turno para elegir A desde aquí, si alguna vez el juego alcanzara este nodo.
- Como A quiere ganar, A se moverá a un nodo hijo con valor máximo.
- En el caso de que x sea un nodo circular, debe estar en un nivel par, y si el nodo alguna vez alcanza ese nodo entonces será el turno para que mueva B.
- Como B no juega a ganar, hará un movimiento que minimice la posibilidad de que gane A. En este caso, la siguiente configuración será la que de el valor

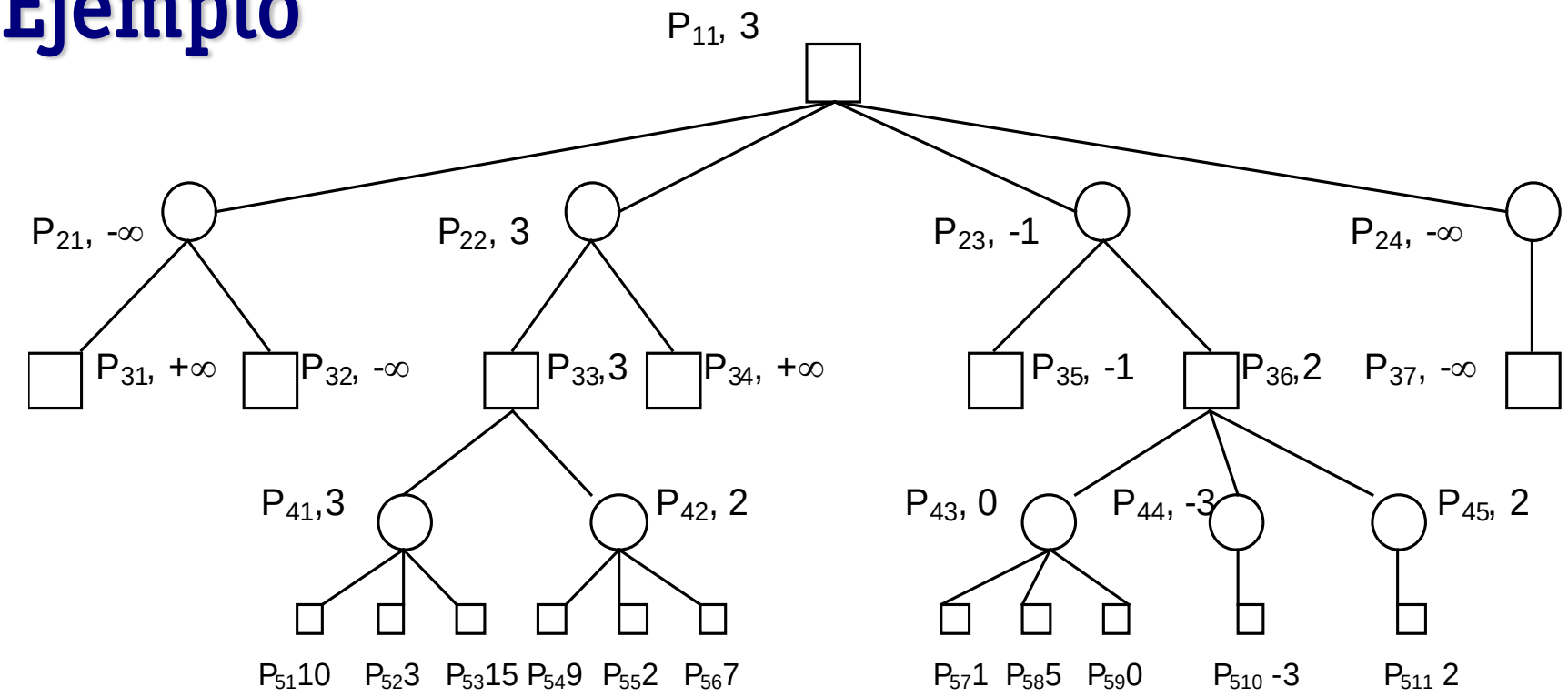
$$\text{Min } \{V[C(i)],\ 1 \leq i \leq d\ \}$$

- La ecuación define lo que se denomina **procedimiento minimax** para determinar el valor de la configuración x.

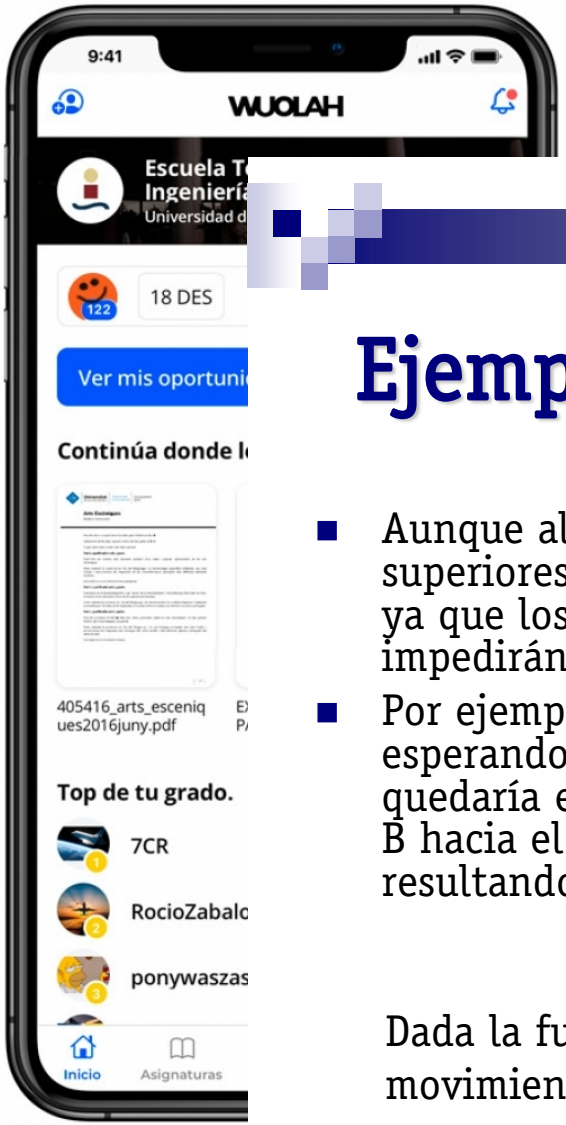
Un ejemplo: Tres en raya



Ejemplo

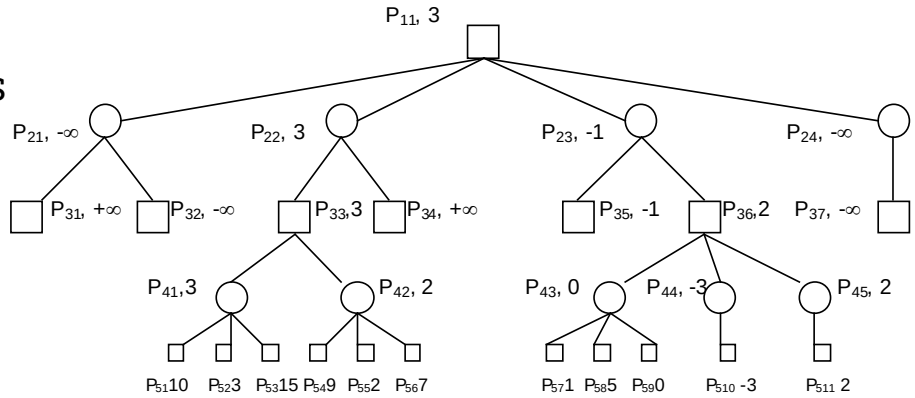


- P(11) representa una configuración arbitraria desde la que A tiene que efectuar un movimiento. Los valores de los nodos hijo se obtienen evaluando la función E(x). El valor de P(11) se obtiene comenzando en los nodos del nivel 4, y usando la ecuación para calcular sus valores.
- El valor resultante para P(11) es 3, lo que significa que partiendo de P(11) lo mejor que A puede esperar es alcanzar una configuración de valor 3.



Ejemplo de desarrollo de un juego

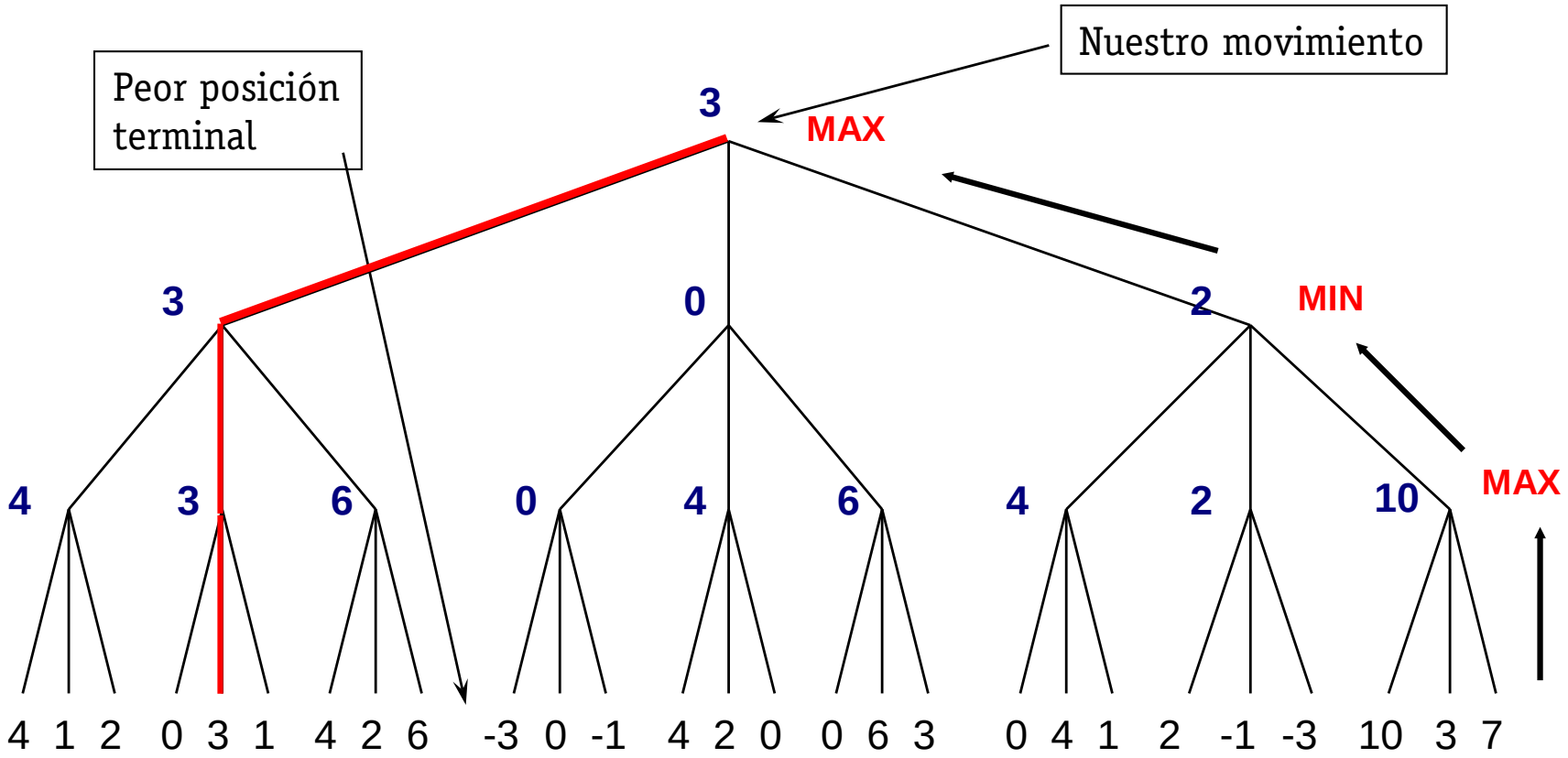
- Aunque algunos nodos tienen valores superiores a 3, nunca serán alcanzados ya que los contra-movimientos de B lo impedirán.
- Por ejemplo, si A se mueve a P(21) esperando ganar el juego en P(31), A quedaría estupefacto cuando viera que B hacia el contramovimiento a P(32), resultando una perdida para A.



Dada la función de evaluación de A y el árbol del juego de la figura, el mejor movimiento de A es hacia la configuración P(22).

Habiendo hecho este movimiento, aún podría ser que el juego no alcanzara la configuración P(52) ya que, en general, B podría usar una función de evaluación distinta, lo que podría proporcionar valores diferentes a las distintas configuraciones.

Otro ejemplo de juego



Desarrollo del juego

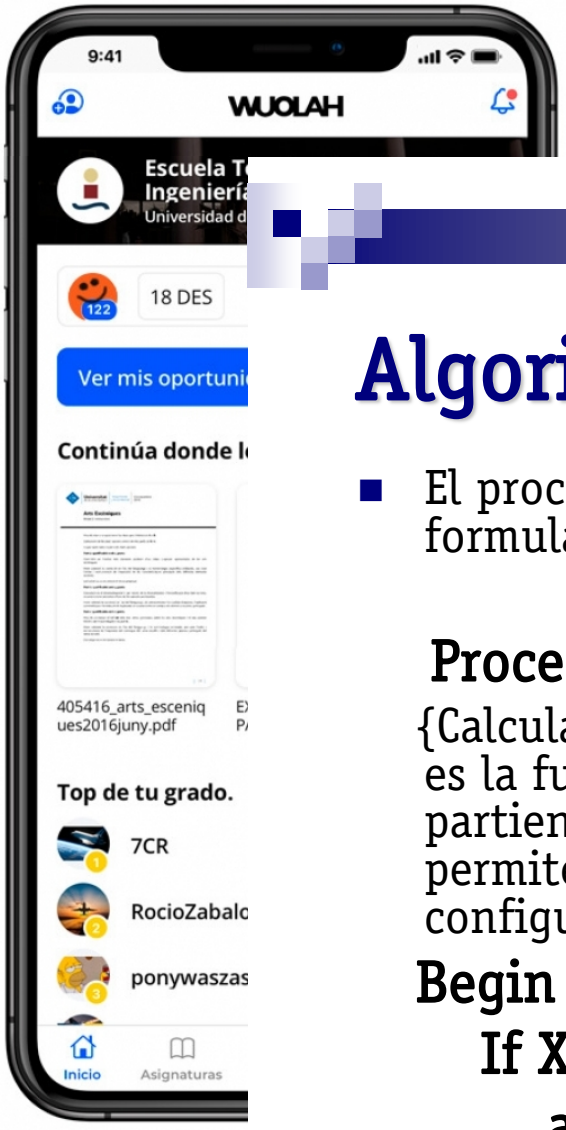
- Este procedimiento no es demasiado interesante en juegos con árboles pequeños. Donde realmente es potente es en juegos, como el ajedrez, en los que el correspondiente árbol es tan grande que es imposible generarlo completamente.
- Se estima que el árbol del ajedrez tiene mas de 10^{100} nodos por lo que, aunque usáramos un ordenador potentísimo, podríamos tardar mas 10 años en generarlo por completo.
- Así en juegos de características similares, la decisión de como efectuar un movimiento se realiza mediante la exploración de unos cuantos niveles.
- La función de evaluación E(X) se usa para obtener los valores de los nodos hoja del subárbol generado

Construcción de un algoritmo

- Intentemos escribir un algoritmo que pueda usar A para calcular V(X).
- El procedimiento para calcular V(X) puede usarse también para determinar el siguiente movimiento que debería hacer A.
- Puede obtenerse un procedimiento recursivo para calcular V(X) usando minimax si disponemos la definición de minimax en la siguiente forma

$$V'(X) = e(X) \quad \text{si X es una hoja del subárbol generado}$$
$$= \text{Max } \{-V'[C(i)],\ 1 \le i \le d\} \quad \text{si X no es una hoja del subárbol generado y los C(i), son los hijos de X}$$

- donde e(X) = E(X) si X es una posición desde la que A se tiene que mover, pero e(X) = -E(X) en caso contrario.
- Esta ecuación calcula V'(X) = V(X) como la ecuación anterior.



Algoritmo

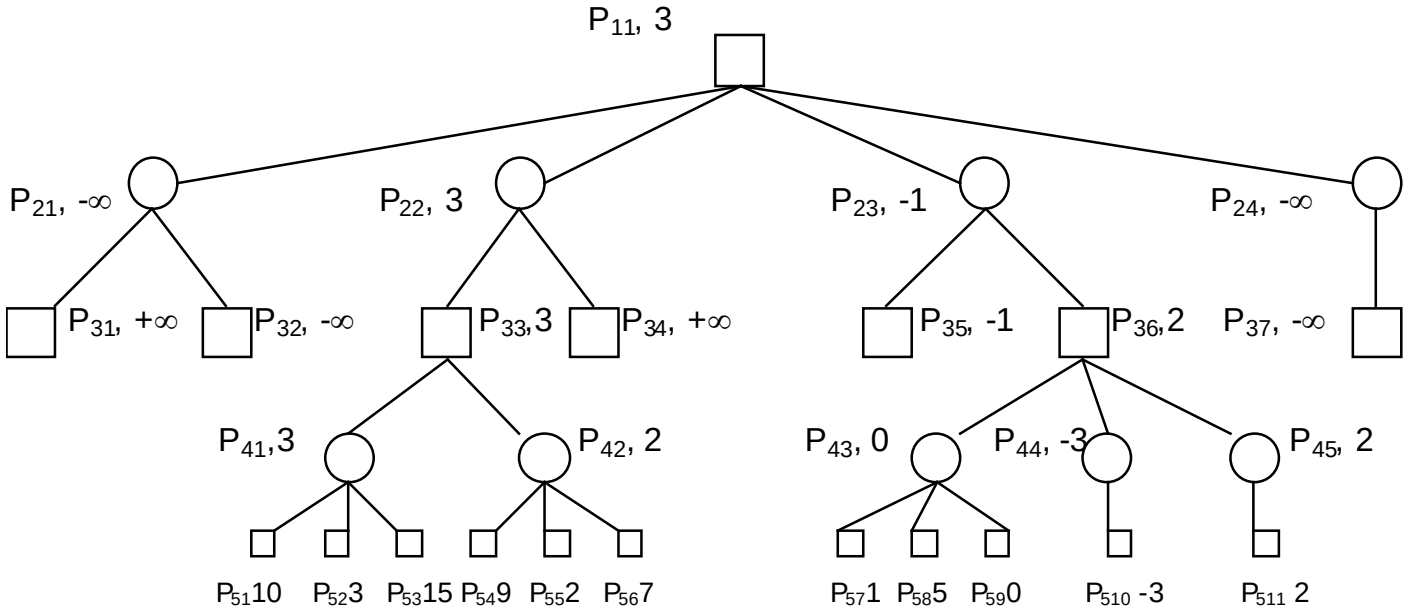
- El procedimiento recursivo para evaluar V'(X) basado en la anterior formula es el siguiente,

Procedimiento VE(X,l)

{Calcula V'(X) realizando a lo sumo l movimientos hacia adelante. e(X) es la función de evaluación de A. Se supone por conveniencia que partiendo de una configuración X, los movimientos legales del juego solo permiten transiciones a las configuraciones C(1),...,C(d) si X no es una configuración terminal}

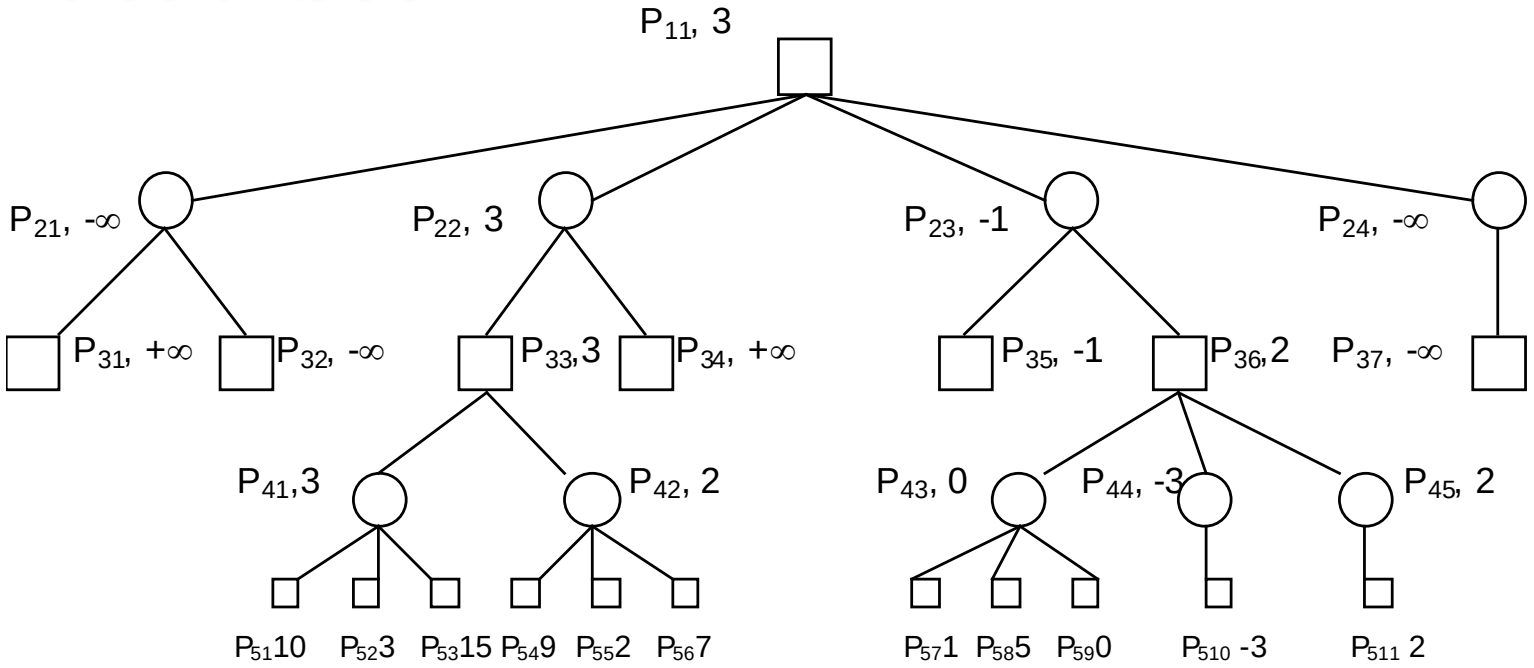
```
Begin
    If X es terminal o l = 0 Then return (e(X))
        ans = -VE(C(1), l-1)
        For i = 2 to d do
                ans = Max(ans, -VE(C(i), l-1))
        Return (ans)
end
```

Funcionamiento del algoritmo



- Sobre el juego de la figura, una llamada a este algoritmo con X = P(11) y l = 4 produciría la generación completa del árbol del juego.
- Los valores de las diversas configuraciones se determinarían en el orden P(31), P(32), P(21), P(51), P(52), P(53), P(41), P(54), P(55), P(56), P(42), ..., P(37), P(24) y P(11).

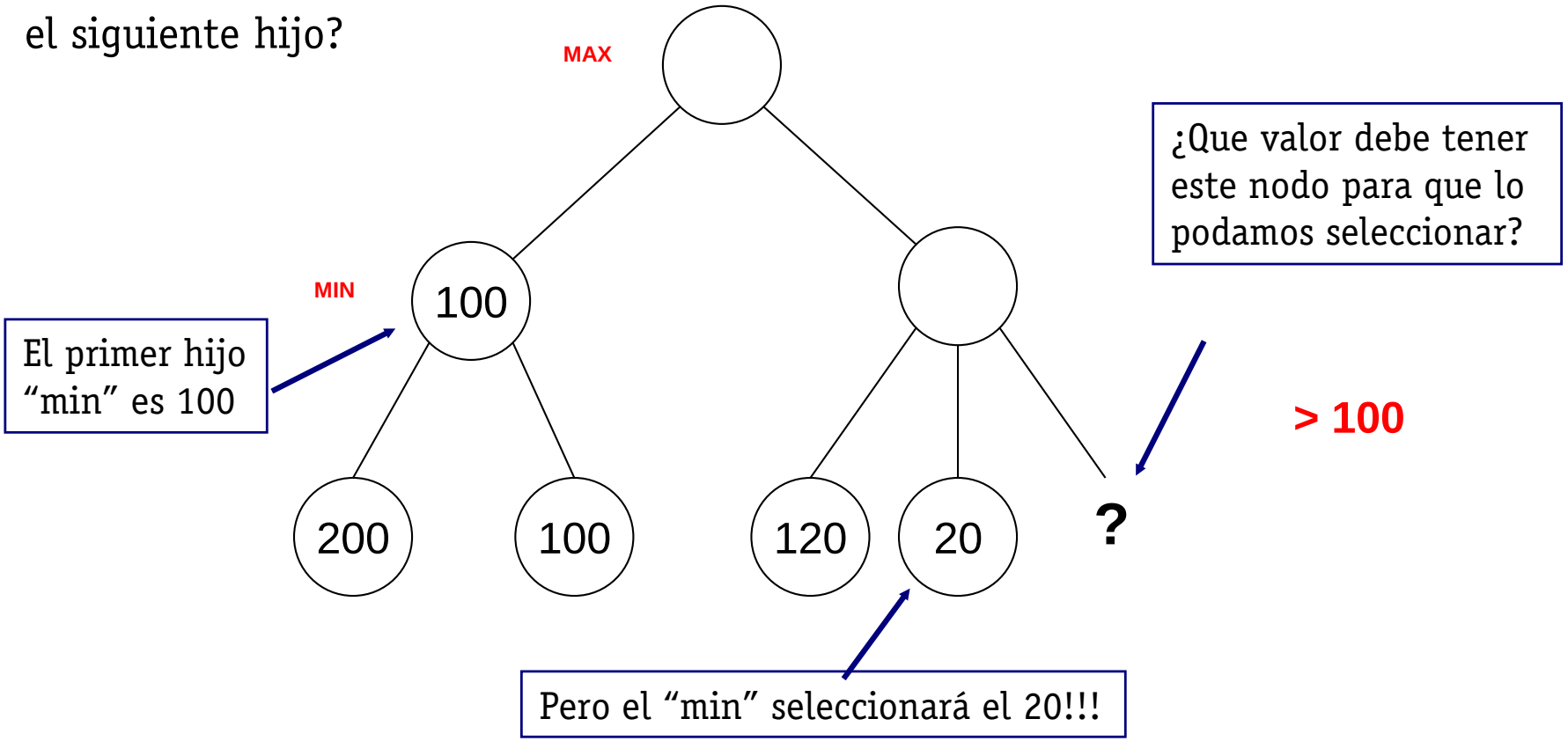
Poda del árbol

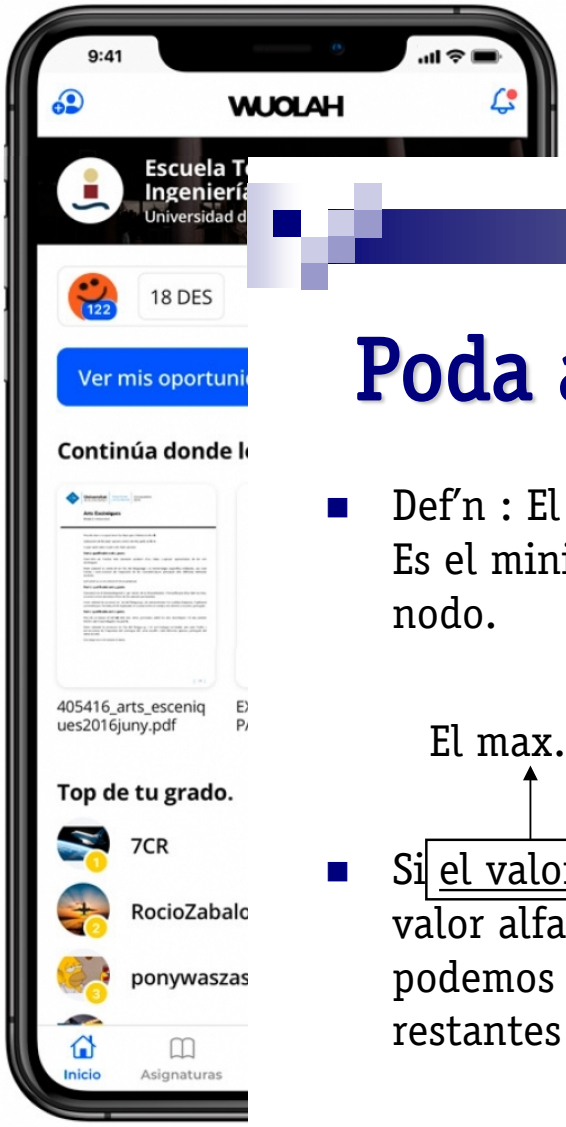


- Después de haber calculado V[P(41)], se sabe que V[P(33)] es al menos V[P(41)] = 3.
- A continuación, cuando se ve que V[P(55)] es 2, sabemos que V[P(42)] es a lo mas igual a 2. Como P(33) es una posición Max, V[P(42)] no puede afectar a V[P(33)].
- Independientemente de los valores de los restantes hijos de P(42), el valor de P(33) no se determina por V[P(42)] ya que V[P(42)] no puede ser mas que V[P(41)].

La poda aún mas clara

Deberiamos evaluar el siguiente hijo?





Poda alfa

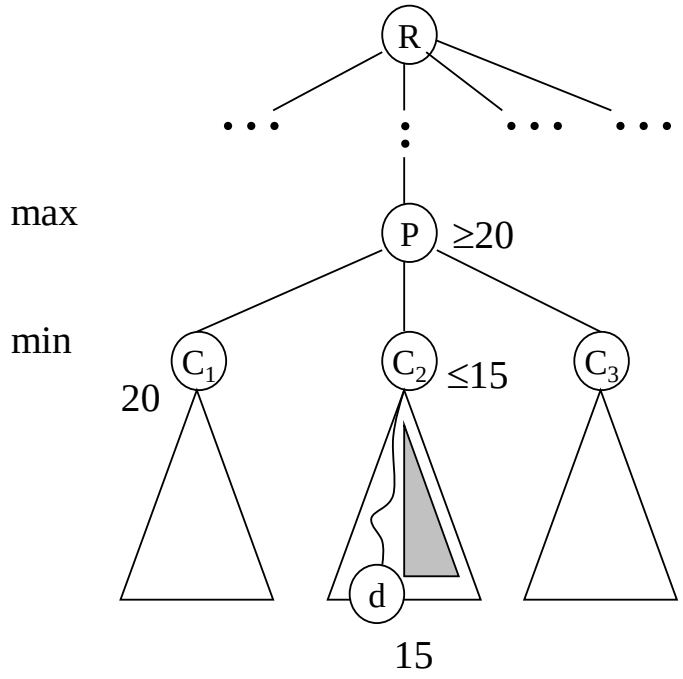
- Def'n : El valor alfa de una nodo max. Es el minimo valor posible para ese nodo.

El max. Valor posible

- Si el valor de un nodo min es ≤ el valor alfa de su padre, entonces podemos parar el examen de los restantes hijos de este nodo min.

Poda alfa !!!

$w(P) = \max\{\ w(C_1), w(C_2), w(C_3)\ \}$
$20 \quad \leq 15$



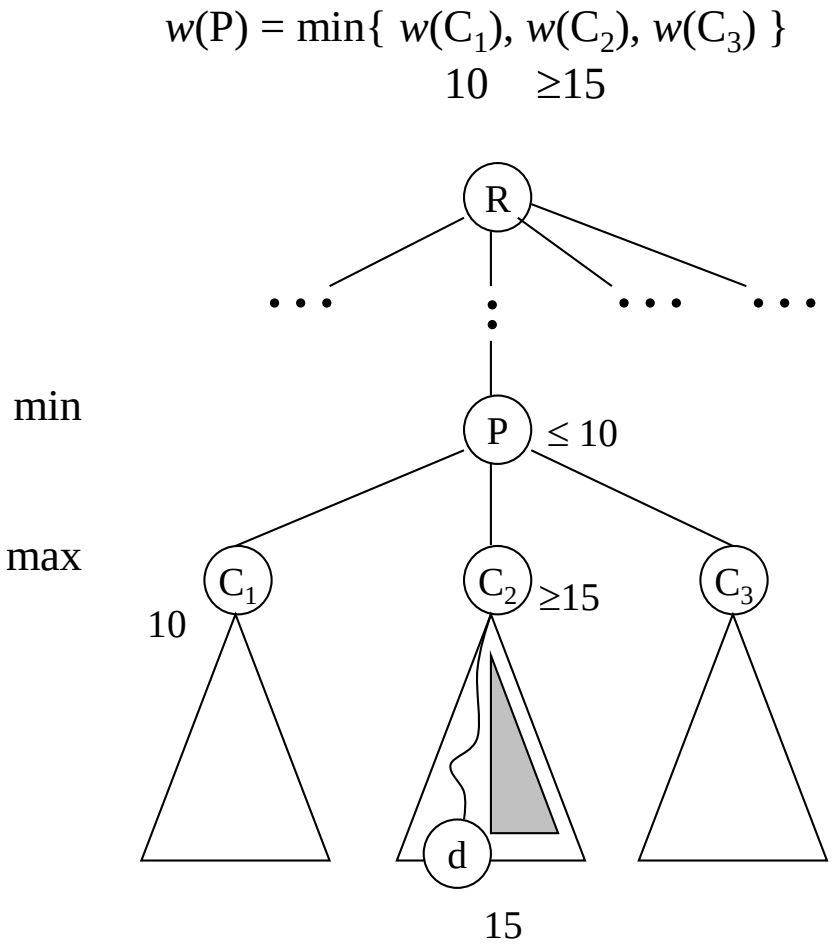
Poda beta

- Def'n : El valor beta de un nodo min. Se define como el maximo Valor posible para ese nodo.

  El min. Valor posible

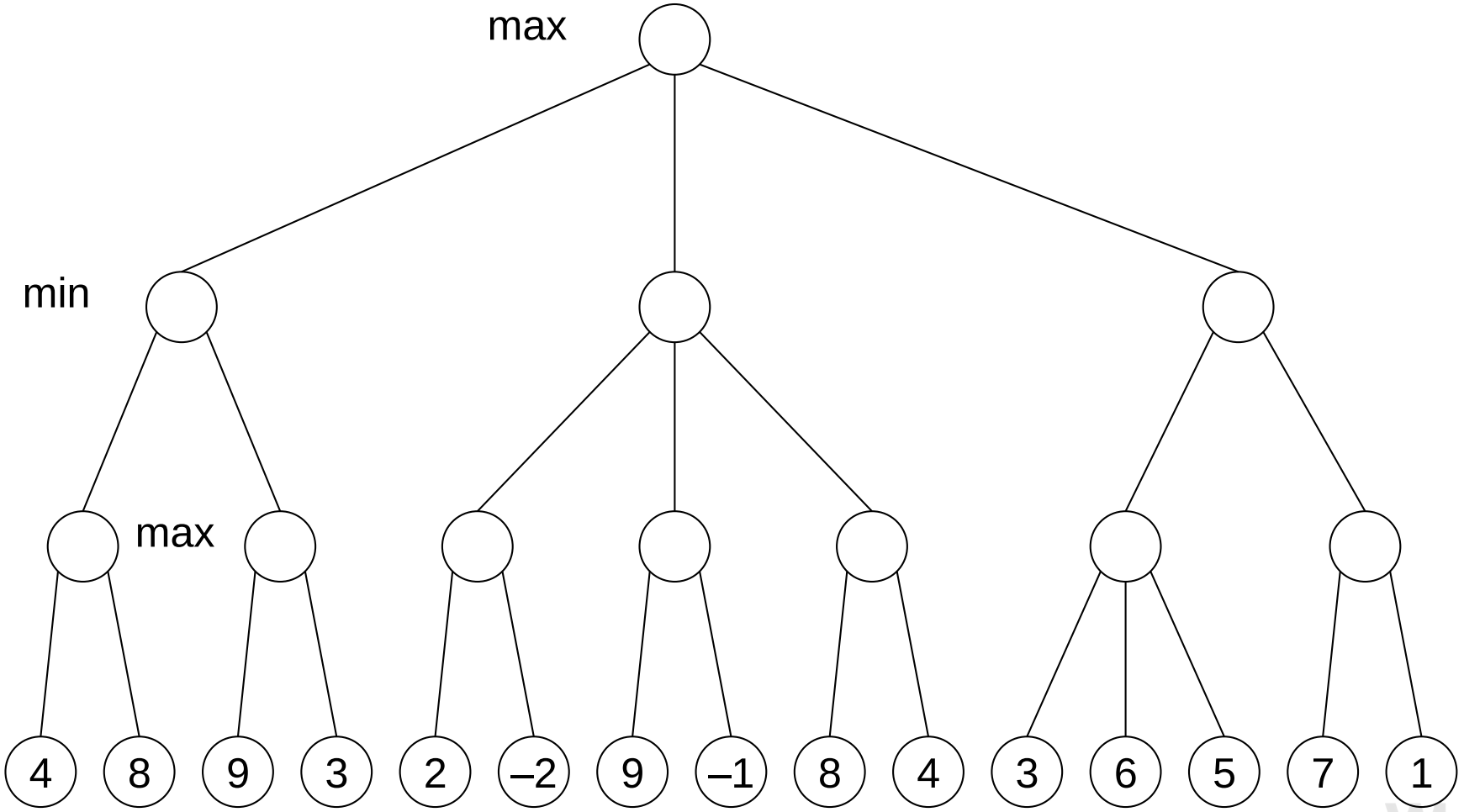
- Si el valor de una nodo max. es ≥ que el valor beta de su padre, entonces podemos parar de examinar los restantes hijos de es nodo max.

  Poda Beta!!!

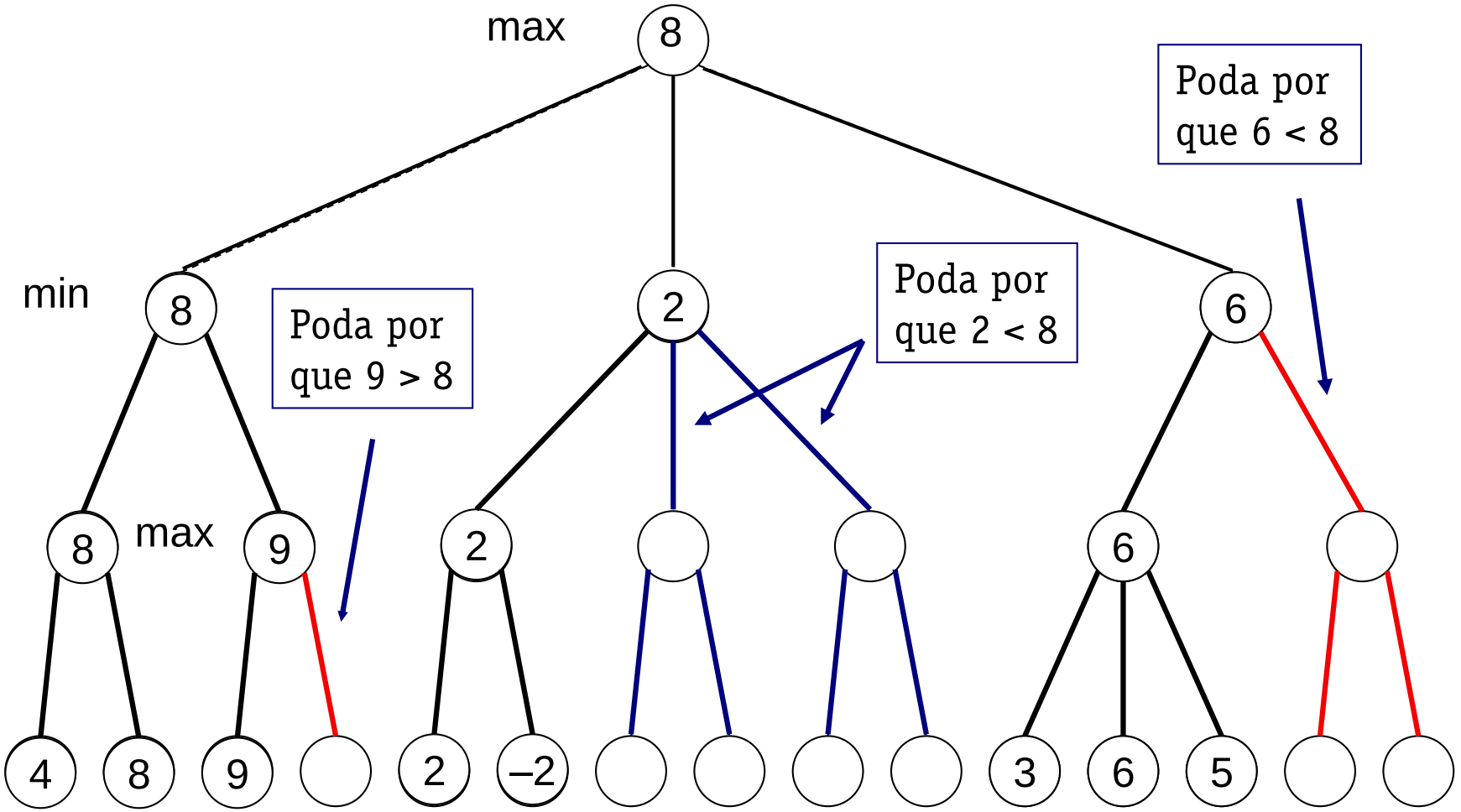


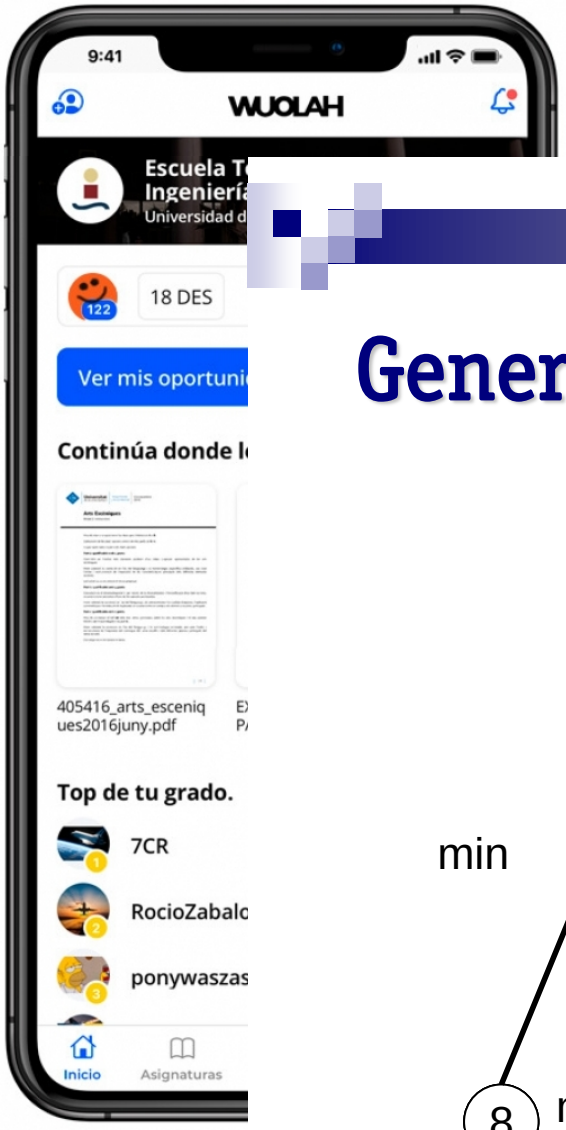
Ejemplo

- Arbol Minimax

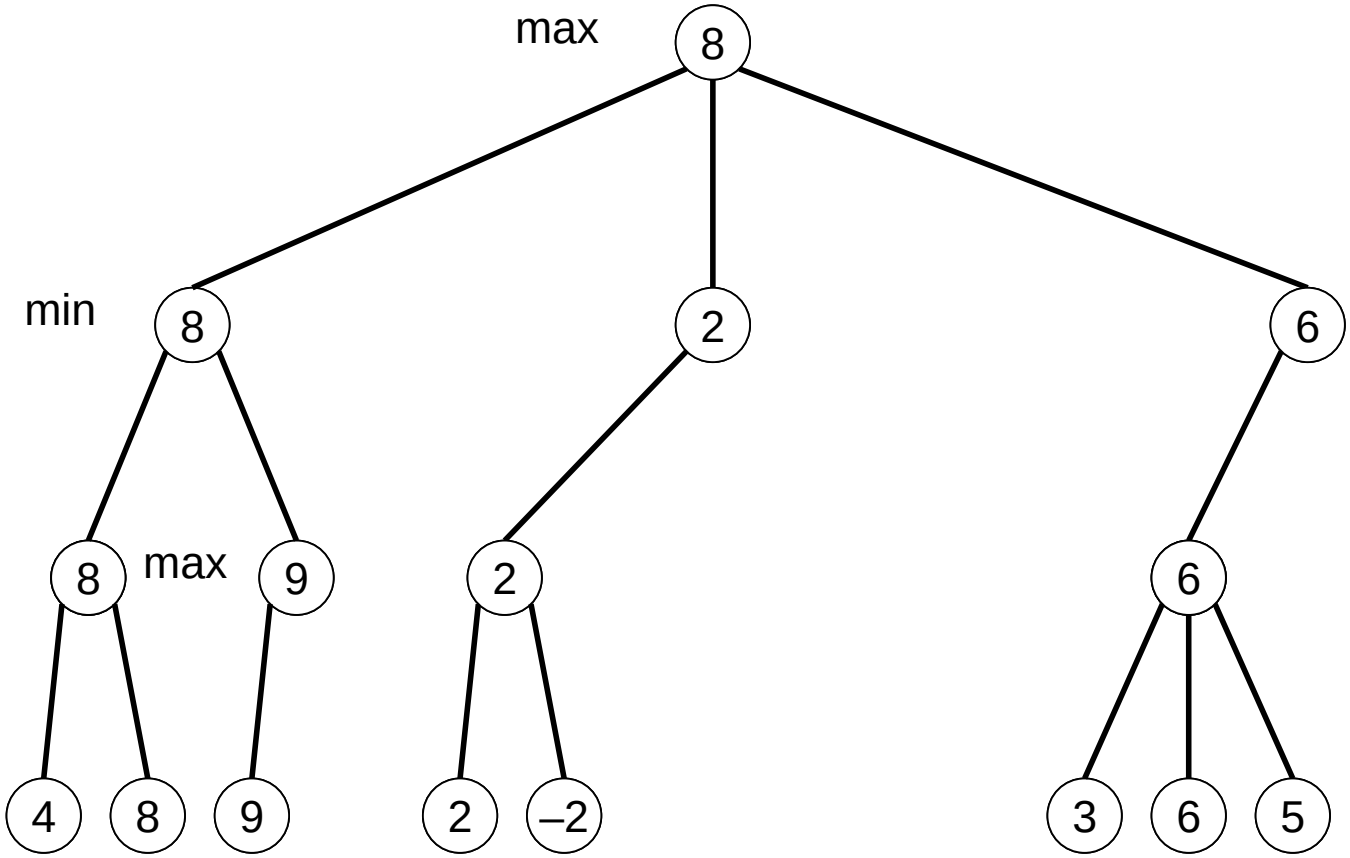


Poda Alfa-Beta





Generacion real



Poda alfa-beta

- Las dos reglas anteriores pueden combinarse juntas para dar lo que se denomina Poda Alfa-Beta.
- Para introducir la poda alfa-beta en el algoritmo VE es necesario establecer esta regla en los términos de la ecuación para V'(x), ya que bajo ese esquema todas las posiciones son posiciones max ya que los valores de las posiciones min se multiplicaron por -1.
- La regla de la poda alfa-beta se reduce entonces a lo siguiente:
- Llamemos B-valor al mínimo valor que una cierta posición puede tener. Para cualquier posición X, sea B el B-valor de su padre y D = -B. Entonces si el valor de X es mayor o igual que D, podemos terminar la generación de los restantes hijos de X.
- La incorporación de esta regla en el algoritmo VE es sencilla, y produce el algoritmo VEB siguiente

Poda alfa-beta

- D es un parámetro adicional D que es el valor negativo del B-valor del padre de X

```
Procedimiento VEB(X,l,D)
 {Determina V'(X) usando la regla B y haciendo solo l movimientos hacia adelante. Las restantes hipótesis y notaciones son las mismas que en el algoritmo VE}
 Begin
        Si X es terminal o l = 0 Entonces return (e(X))
        Temp = -VEB(C(1), l-1, ∞)
        Para i = 2 hasta d hacer
            Si Temp ≥ D entonces return (Temp)
        Temp = Max(Temp, -VEB(C(i), l-1, -Temp))
        Return (Temp)
 end
```