Sistemas Inteligentes I

Tema 3. Juegos

José A. Montenegro Montes
monte@lcc.uma.es

Resumen

- Juegos
- Algoritmo Minimax
- Poda Alfa-Beta
- Funciones de Evaluación

Juegos

- Entornos multiagente, donde cada agente debe considerar las acciones de los otros agentes.
- **Juegos**: Entornos competitivos donde los objetivos del agente están en conflicto, dan lugar a problemas de búsqueda entre adversarios.
  - Ajedrez, Otelo, Backgammon, Go
  - Ajedrez: Árbol de búsqueda tiene 10^{154} nodos.
- Capacidad de tomar decisión cuando no es factible calcular la decisión óptima.
  - **Poda**: Nos permiten ignorar partes del árbol búsqueda.
  - **Funciones evaluación**: Heurísticas que permiten aproximar la utilidad sin hacer búsqueda completa.

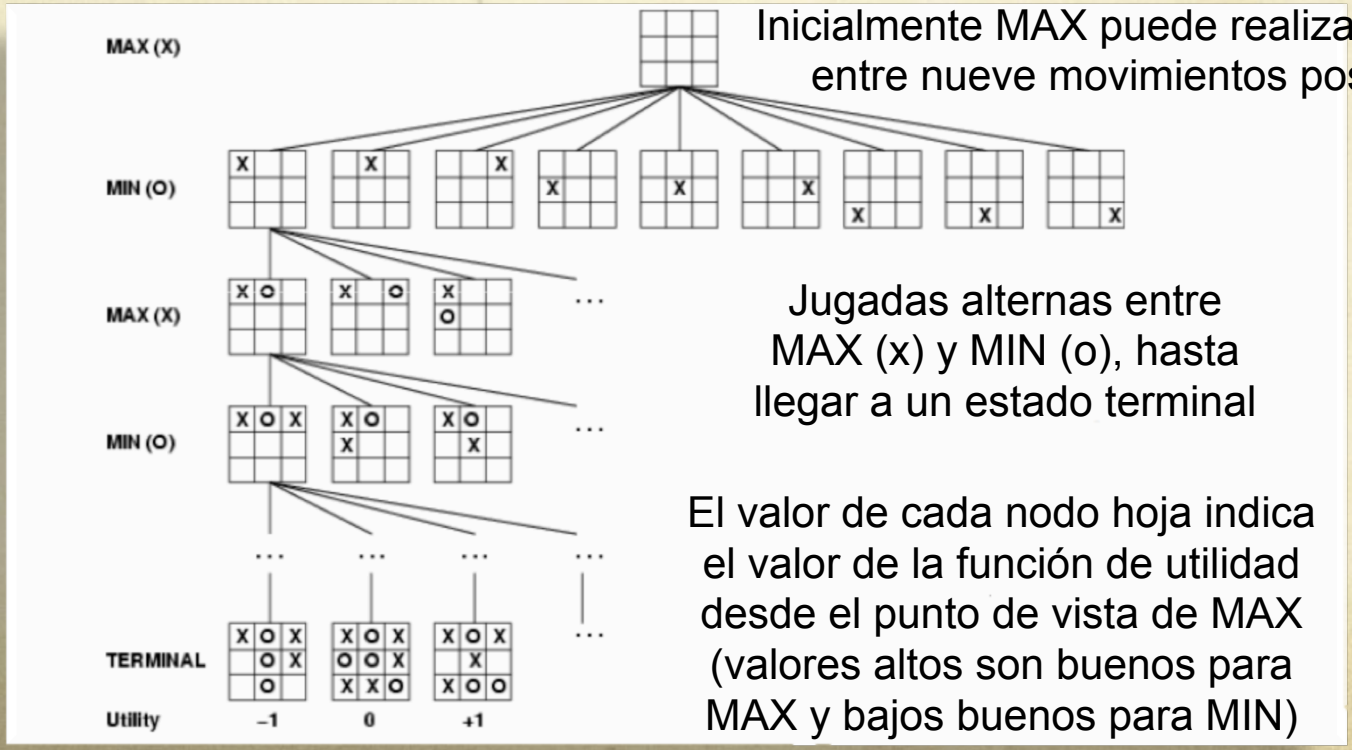
Componentes Juegos

- Juegos, clase de problemas de búsqueda:
  - **Estado Inicial**: Posición tablero y jugador que mueve
  - **Función sucesor**: Lista pares (movimiento, estado) , estado resultante.
  - **Test terminal**: Cuando finaliza el juego. Estados terminales.
  - **Función Utilidad**: Valor numérico a los estados terminales.
    - Por ejemplo (suma nula): Triunfo +1, Pérdida -1, Empate 0
- **Árbol del juego**: Definido mediante estado inicial y los movimientos legales.

Algoritmo Minimax

Algoritmo Minimax

- Juegos 2 jugadores (MAX y MIN)
  - Primero mueve MAX y luego MIN por turnos hasta que termina
- Árbol del juego



Inicialmente MAX puede realizar uno de entre nueve movimientos posibles

Jugadas alternas entre MAX (x) y MIN (o), hasta llegar a un estado terminal

El valor de cada nodo hoja indica el valor de la función de utilidad desde el punto de vista de MAX (valores altos son buenos para MAX y bajos buenos para MIN)

Algoritmo Minimax

- Tiene por objetivo decidir un movimiento para MAX.
- **HIPÓTESIS**
  - Jugador MAX trata de maximizar su beneficio (función de utilidad).
  - Jugador MIN trata de minimizar su pérdida.
  - Suponemos Jugadores juegan de forma óptima.
- **Aplicación algoritmo:**

  1) Generar árbol entero hasta nodos terminales

  2) Aplicar función *utilidad* a nodos terminales

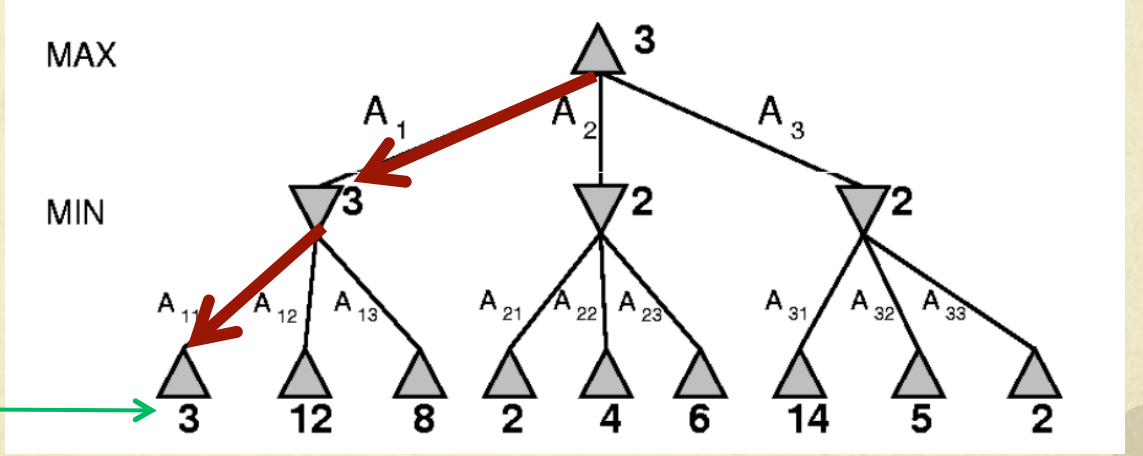
  3) Propagar hacia arriba para generar nuevos valores de *utilidad* para todos los nodos
  - minimizando para MIN
  - Maximizando para MAX

  4) Elección jugada con máximo valor de *utilidad*

Algoritmo Minimax

Valor-MINIMAX(n) =

- UTILIDAD(n) — si n es estado terminal
- max VALOR-MINIMAX(s) — si n es un estado MAX
- min VALOR-MINIMAX(s) — si n es un estado MIN



Utilidad para MAX

Algoritmo Minimax

```
function MINIMAX-DECISION(state) returns una acción
    inputs: state, estado actual en el juego
    v ← MAX-VALUE(state)
    return una acción de SUCCESSORS(state) con valor v
```

```
function MAX-VALUE(state) returns valor utilidad
    if TERMINAL-TEST(state) then return UTILITY(n)
    v ← -∞
    for s en SUCCESSORS(state) do
        v ← MAX(v, MIN-VALUE(s))
    return v
```

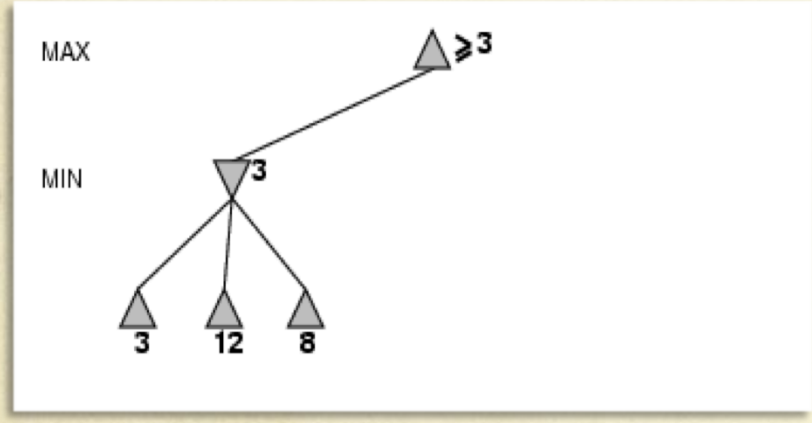
```
function MIN-VALUE(state) returns valor utilidad
    if TERMINAL-TEST(state) then return UTILITY(n)
    v ← ∞
    for s en SUCCESSORS(state) do
        v ← MIN(v, MAX-VALUE(s))
    return v
```

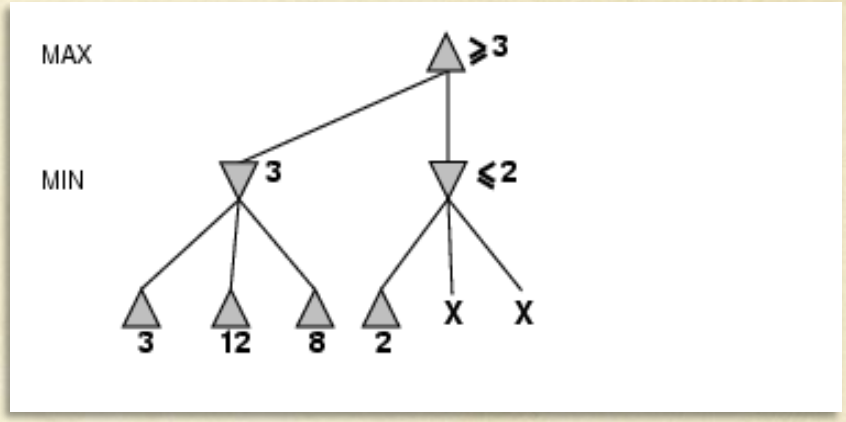
Poda Alfa-Beta

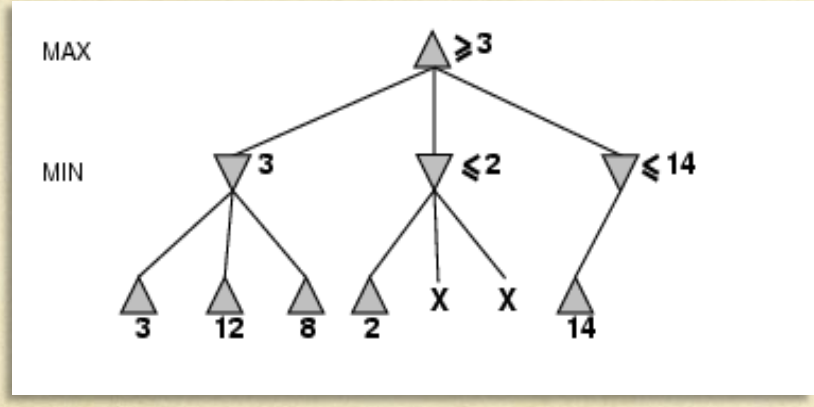
Poda Alfa-Beta

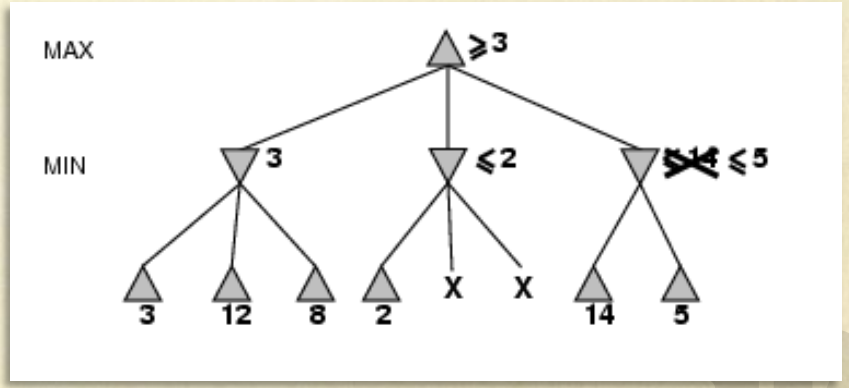
- No afecta al resultado final
- Su efectividad depende del orden en que se consideren los movimientos
- α → valor de la mejor elección para MAX (valor más alto) que hemos encontrado en el camino hasta ahora
- β → valor de la mejor elección para MIN (valor más bajo) que hemos encontrado en el camino hasta ahora

Poda Alfa-Beta

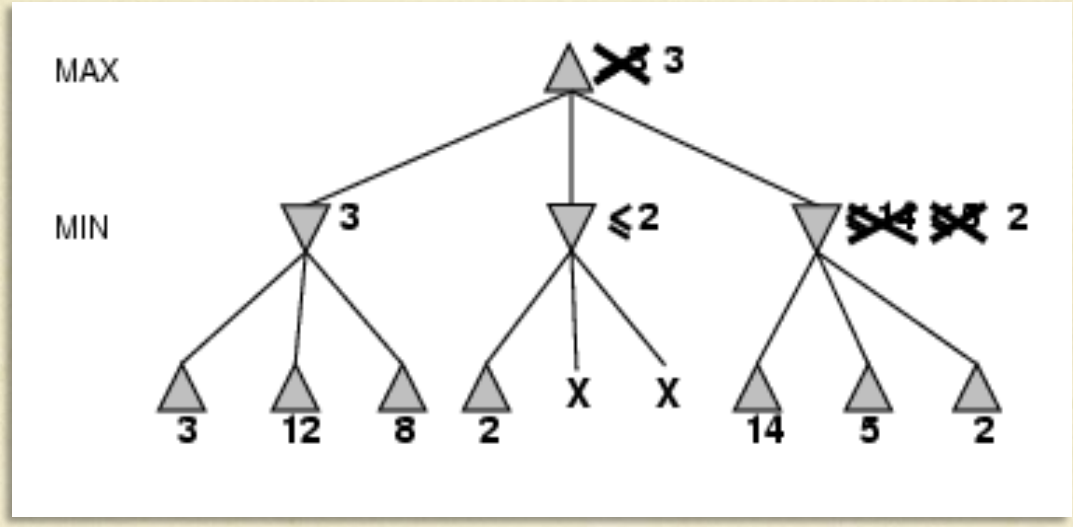








Poda Alfa-Beta



Poda Alfa-Beta

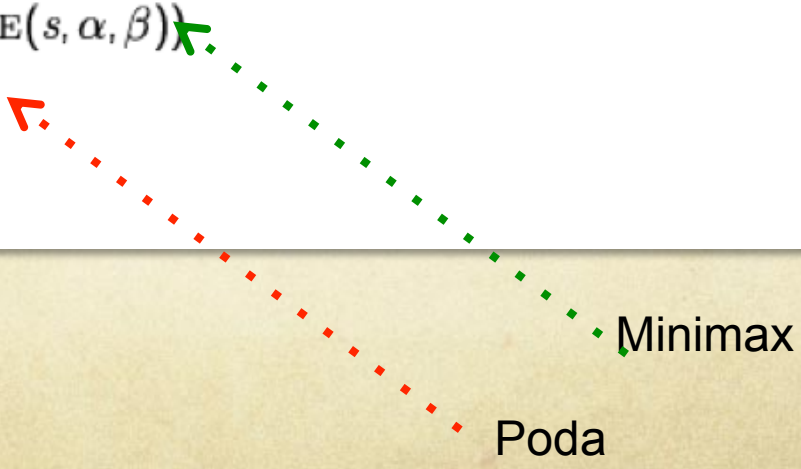
Alfa

Beta

function ALPHA-BETA-SEARCH(*state*) **returns** *an action*
  inputs: *state*, current state in game
  $v \leftarrow$ MAX-VALUE(*state*, $-\infty, +\infty$)
  return the *action* in SUCCESSORS(*state*) with value v

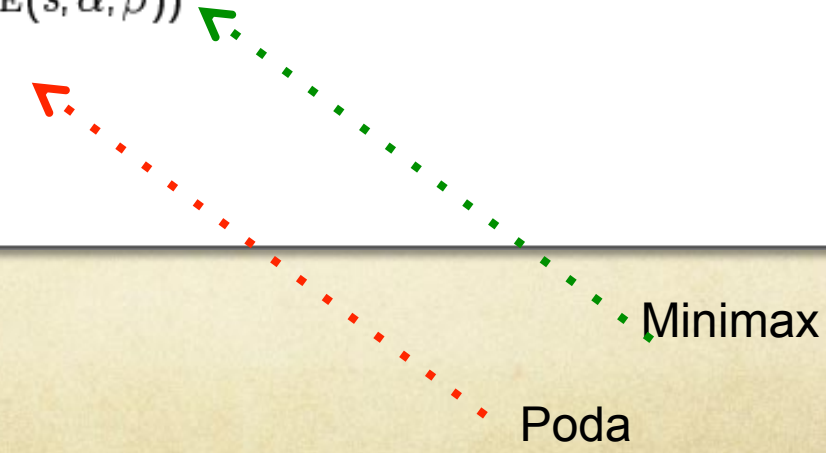
Poda Alfa-Beta

```
function MAX-VALUE(state, α, β) returns a utility value
  inputs: state, current state in game
          α, the value of the best alternative for MAX along the path to state
          β, the value of the best alternative for MIN along the path to state

  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← −∞
  for a, s in SUCCESSORS(state) do
    v ← MAX(v, MIN-VALUE(s, α, β))
    if v ≥ β then return v
    α ← MAX(α, v)
  return v
```

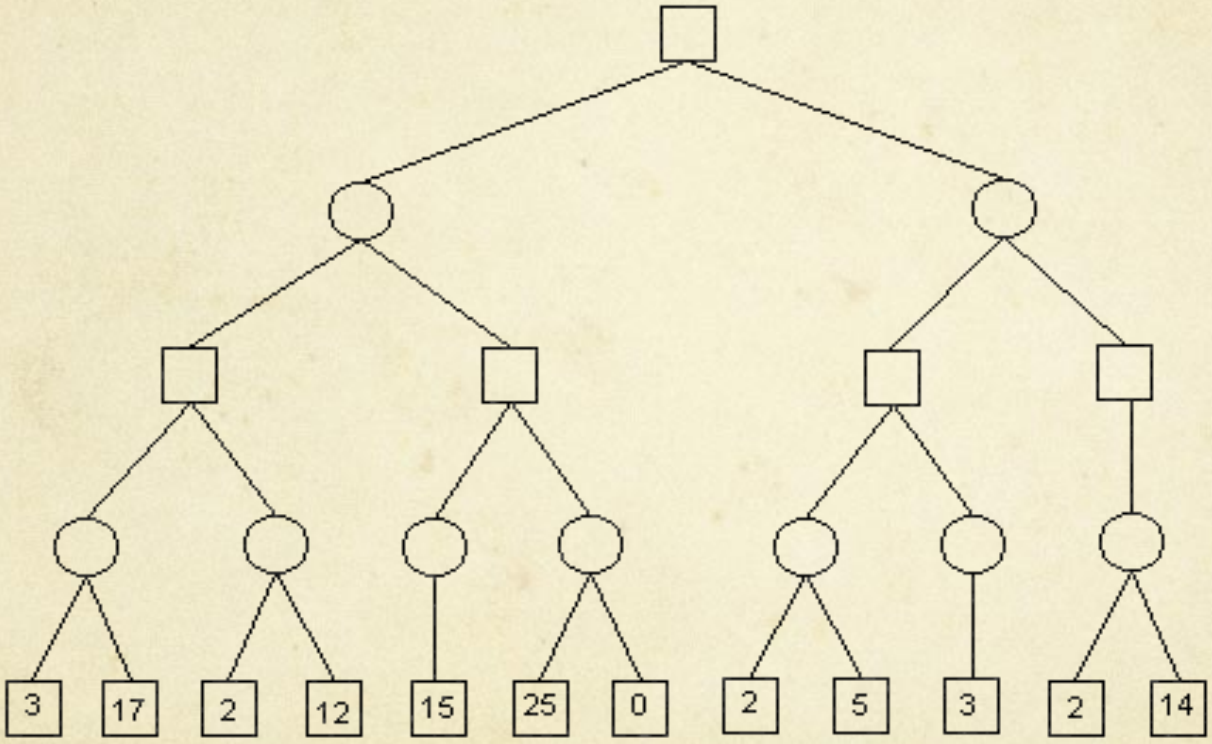


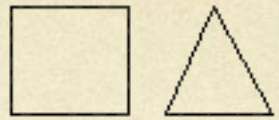
Poda Alfa-Beta

```
function MIN-VALUE(state, α, β) returns a utility value
  inputs: state, current state in game
          α, the value of the best alternative for MAX along the path to state
          β, the value of the best alternative for MIN along the path to state

  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← +∞
  for a, s in SUCCESSORS(state) do
    v ← MIN(v, MAX-VALUE(s, α, β))
    if v ≤ α then return v
    β ← MIN(β, v)
  return v
```

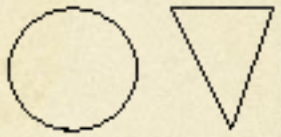


Ejemplo seguimiento Algoritmo



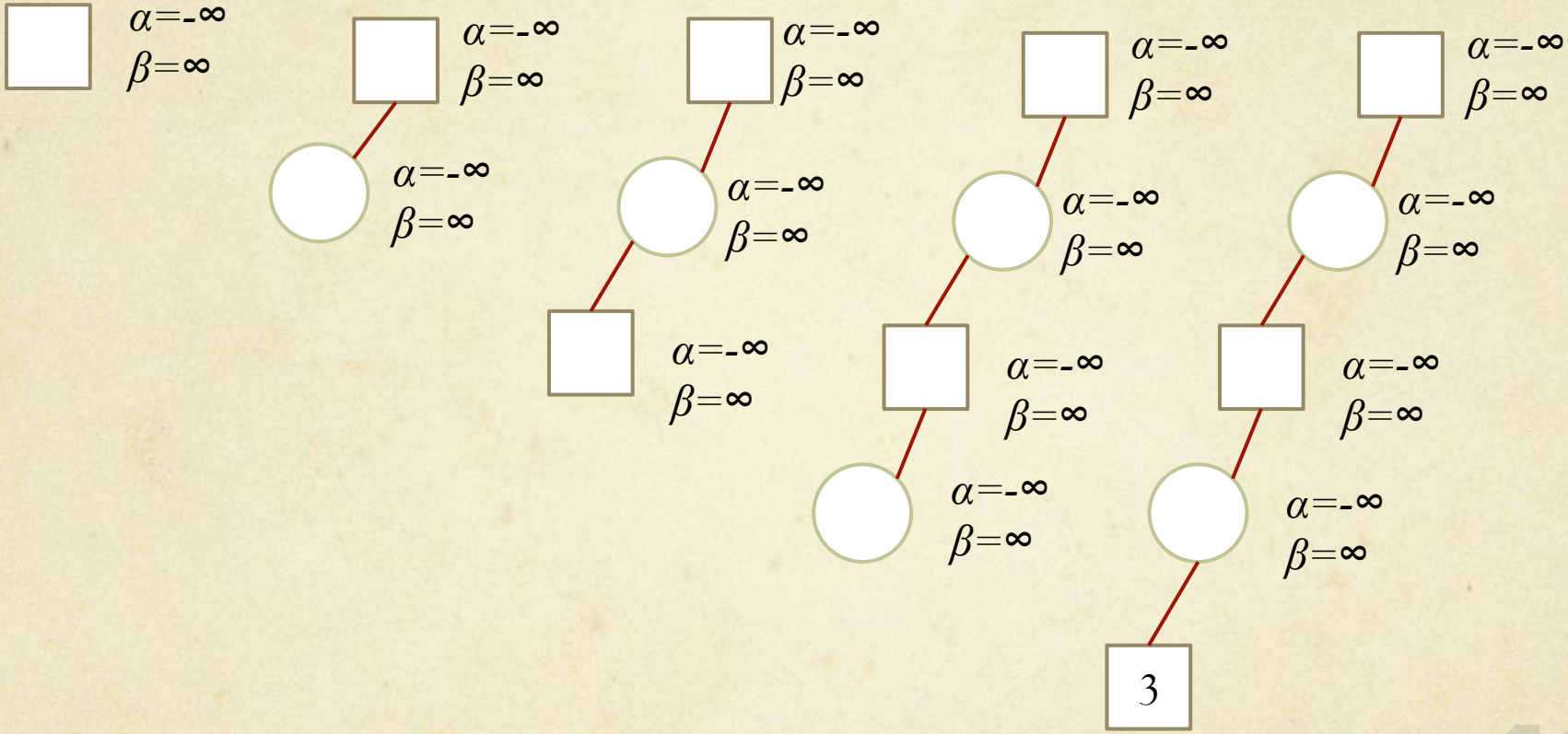


MAX

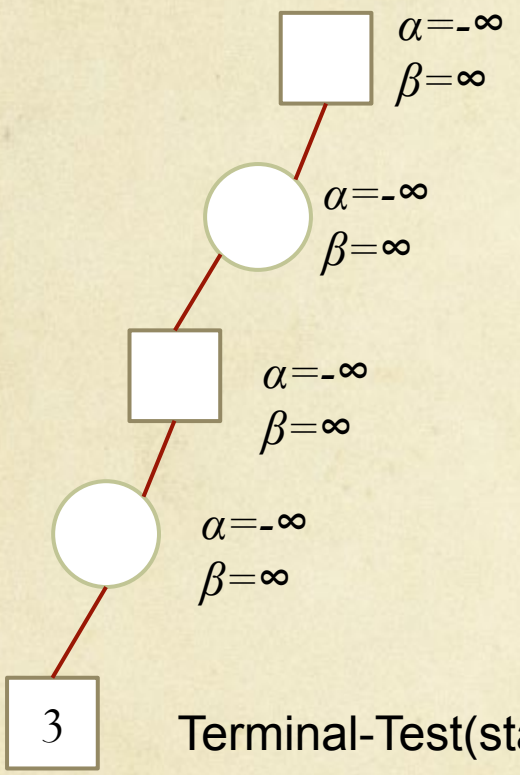


MIN

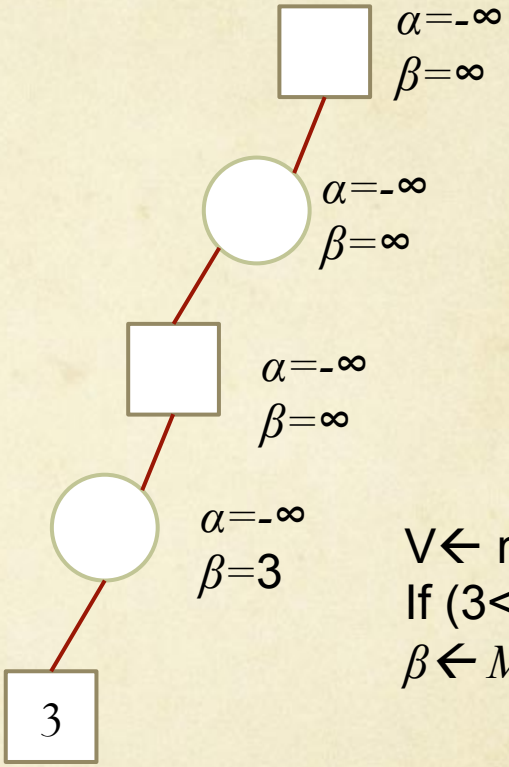
Poda Alfa-Beta



Poda Alfa-Beta

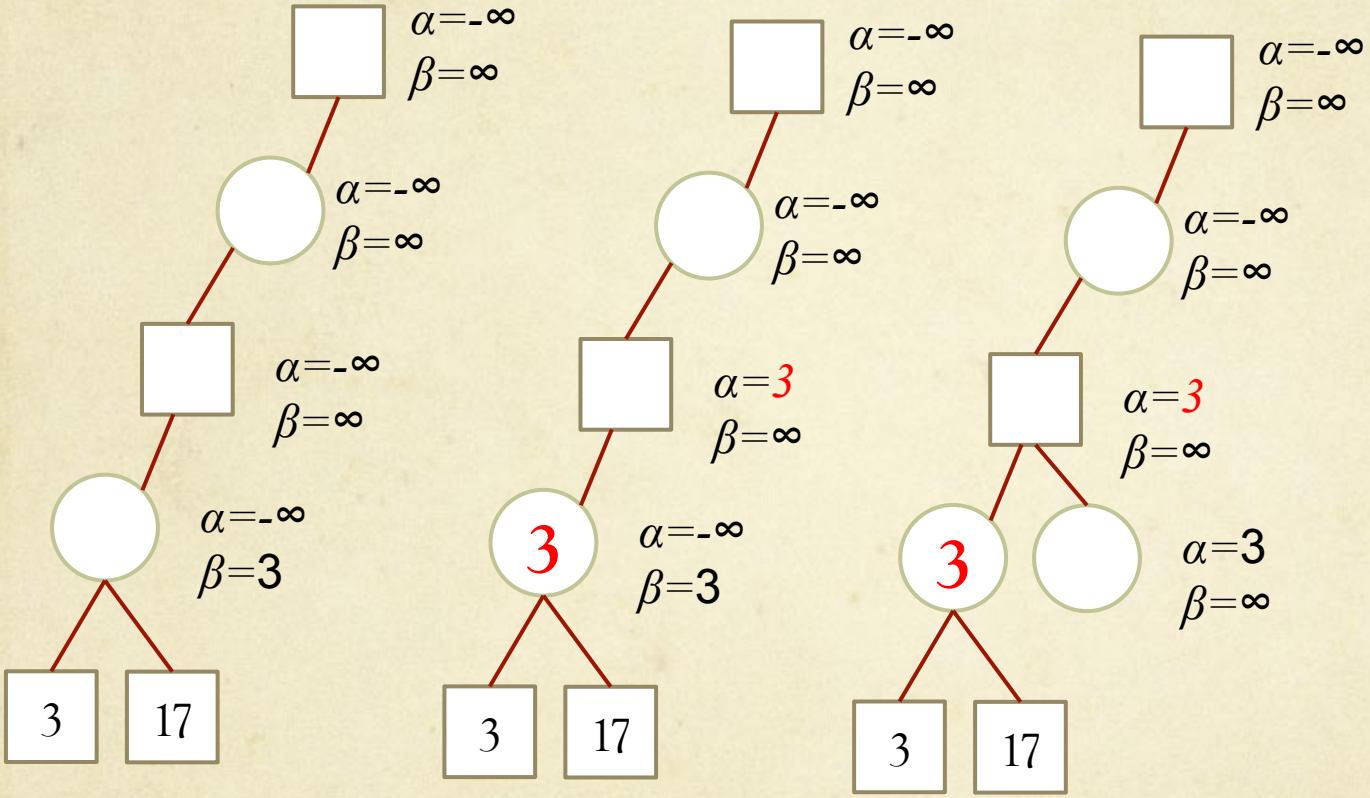


Terminal-Test(state) return 3

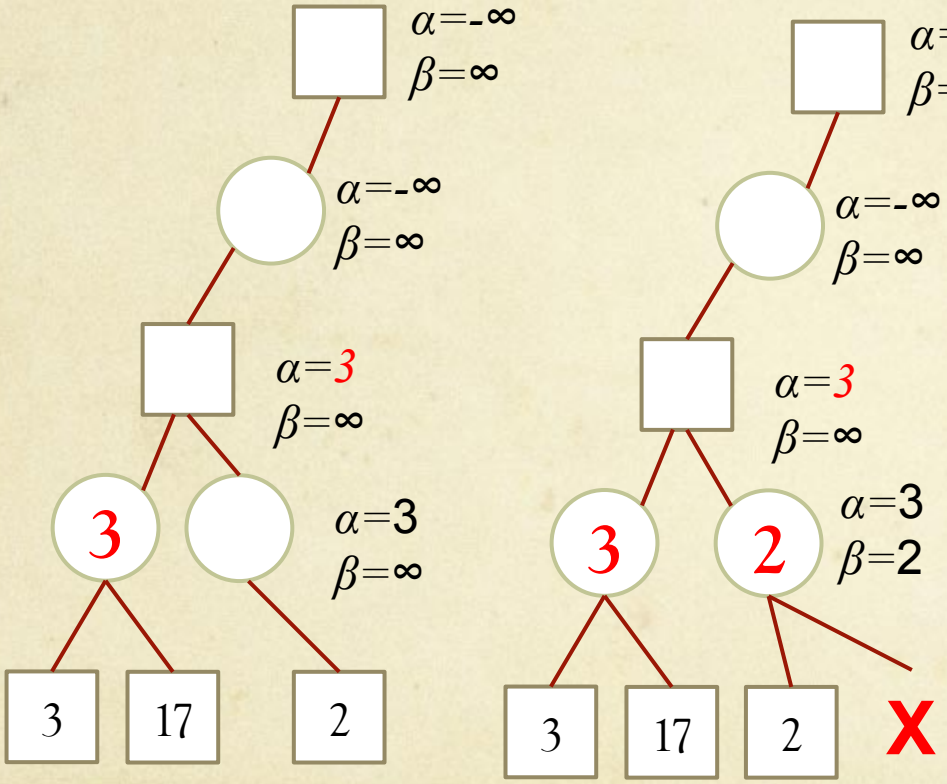


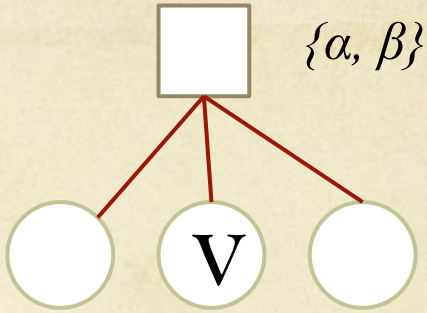
V← min(∞,3)
If (3< -∞) return 3
β← *MIN(*∞,3)

Poda Alfa-Beta

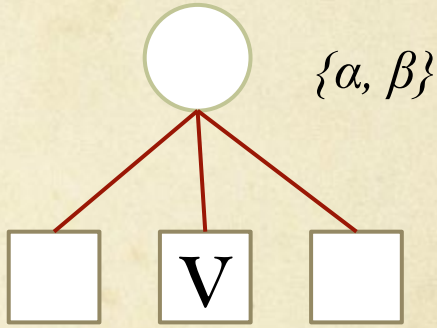


Poda Alfa-Beta





Si V ≥ β poda β



Si V ≤ α poda α

http://cs.ucla.edu/~rosen/161/notes/alphabeta.html

Funciones de Evaluación

Funciones de Evaluación

- **Minimax** genera el espacio de búsqueda **entero**, mientras que el algoritmo **alfa-beta** podamos **partes**.
  - Necesario llegar a estados terminales.
- Trabajo shannon (1950) *programming a computer for playing chess*, propone una función de evaluación heurística a los estados convirtiendo nodos no terminales a terminales.
  - Sustituye **función utilidad** por **función evaluación heurística**, que da una **estimación** de utilidad de la posición y establecemos un test **límite** que decide cuando terminar de aplicar función.
- Su cálculo ha de ser poco costoso

Decisiones imperfectas en juegos de dos adversarios

- Decisión imperfecta: Decisión tomada por el algoritmo minimax sobre un horizonte que no alcanza el final del juego (se asume) y con función de evaluación estimada f = û.

- Función de evaluación, ejemplos:
  - f(n)=
    1) si "n" no es terminal: (número de filas, columnas o diagonales libres para MAX) - (número de filas, columnas o diagonales libres para MIN)
    2) si gana MAX: ∞
    3) si gana MIN: -∞

X		
O		

Nº pos max = 5
Nº pos min = 5

f(n)= 5 - 5 = 0

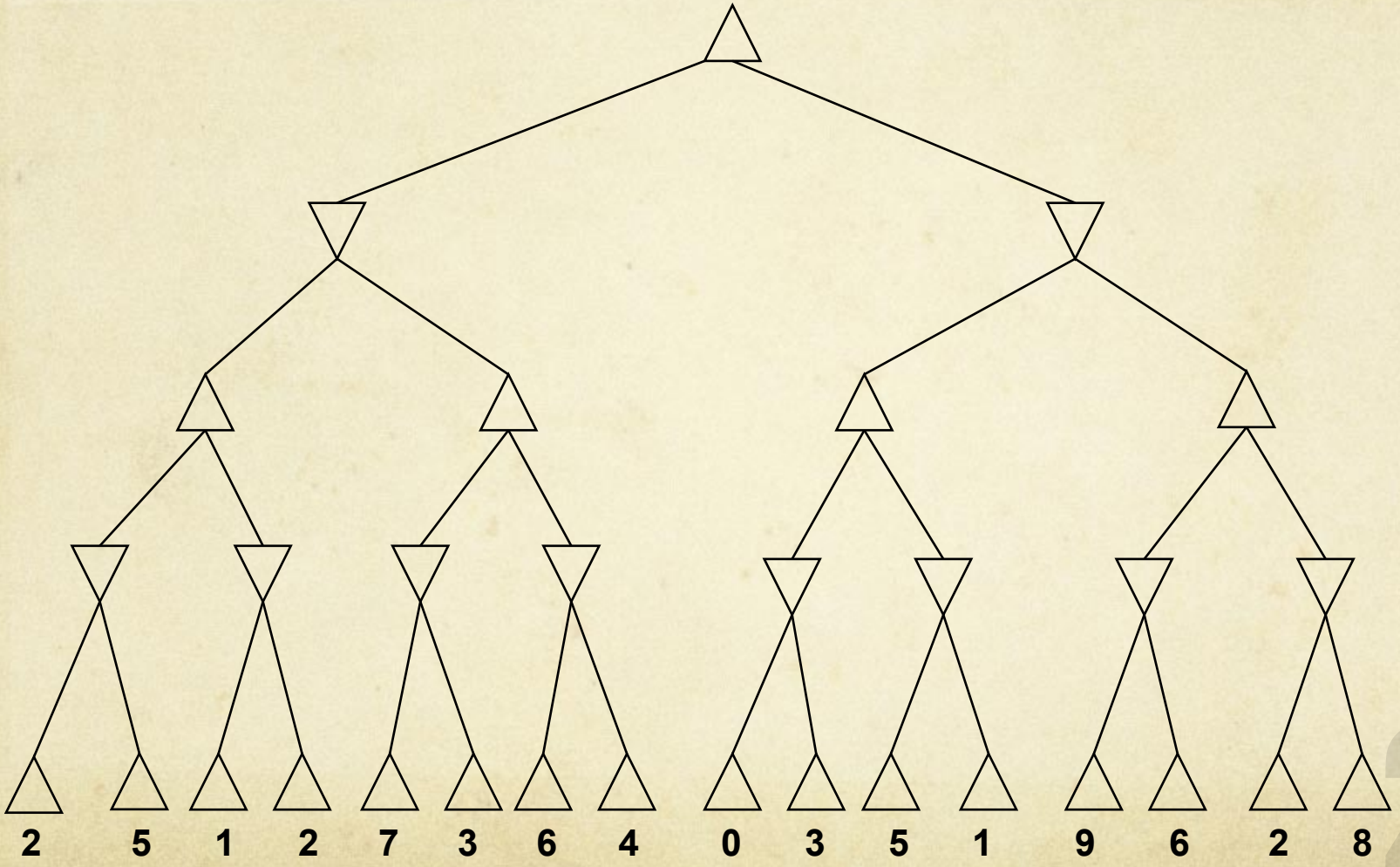
X		
	O	

Nº pos max = 4
Nº pos min = 5

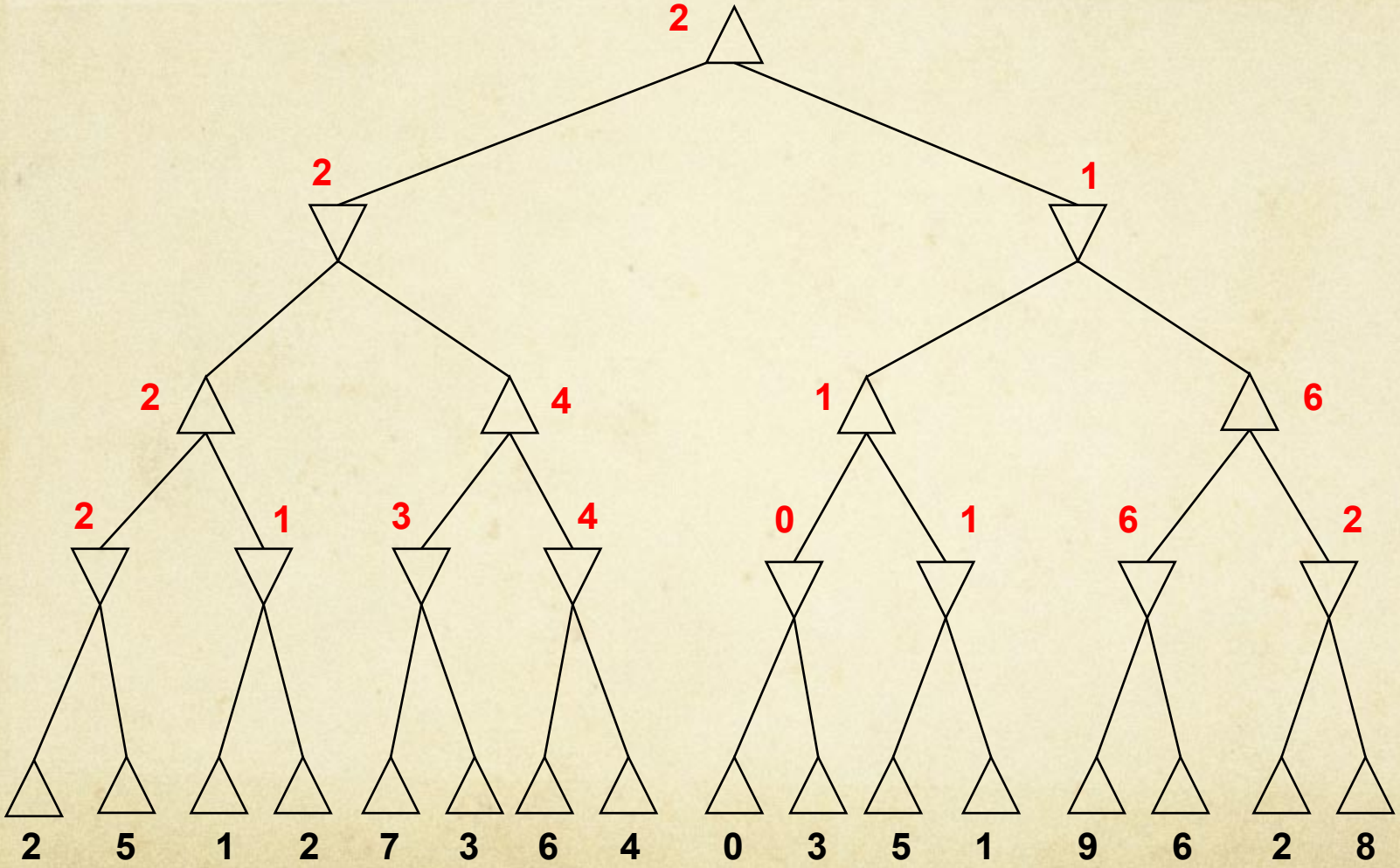
f(n)= 4 - 5 = -1

Ejemplos

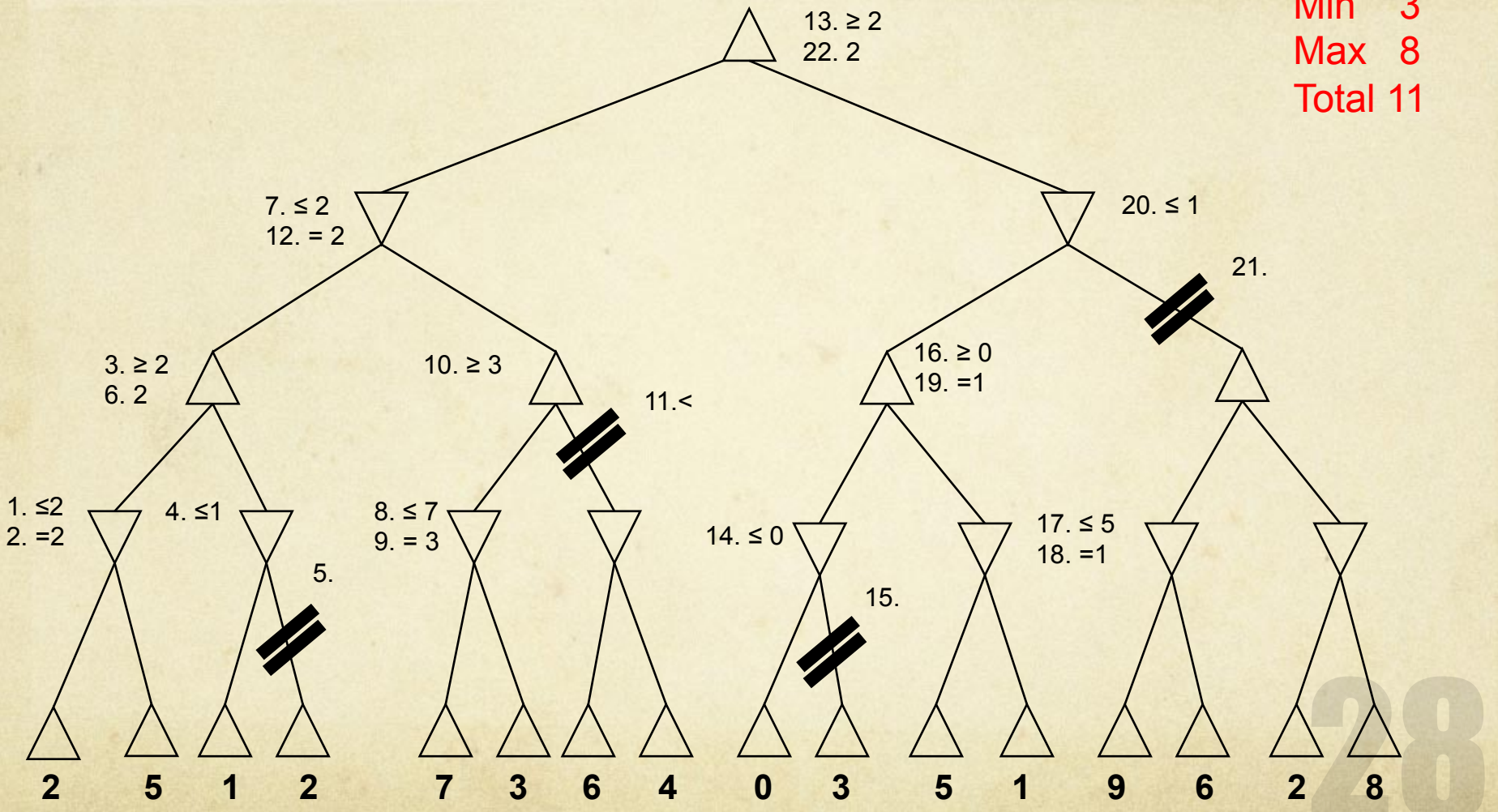
Ejemplo 1



Ejemplo 1 minimax



Ejemplo 1 alfa beta



Sistemas Inteligentes

José A. Montenegro Montes

monte@lcc.uma.es