Metodología y Tecnología de la Programación

Curso 2007-2008

Esquemas algorítmicos. Juegos de Antagonismo

Yolanda García Ruiz	**D228**	**ygarciar@fdi.ucm.es**
Jesús Correas	**D228**	**jcorreas@fdi.ucm.es**

Departamento de Sistemas Informáticos y Computación
Universidad Complutense de Madrid

(elaborado a partir de notas de S. Estévez y R. González del Campo)

Bibliografía

- **Importante**: Estas transparencias son un material de apoyo a las clases presenciales y no sustituyen a la bibliografía básica ni a las propias clases presenciales para el estudio de la asignatura
- Bibliografía básica:
  - [GC01][1]: capítulo 7
- Bibliografía complementaria:
  - [BB97]: capítulo 9 (apartado 9.8)

(1) [GC01] D. Giménez Cánovas *Apuntes y problemas de algorítmica*, Universidad de Murcia, 2001. Disponible en
`http://servinf.dif.um.es/~domingo/apuntes/Algoritmica/apuntes.pdf`

Esquemas algorítmicos. Juegos de Antagonismo

Características generales

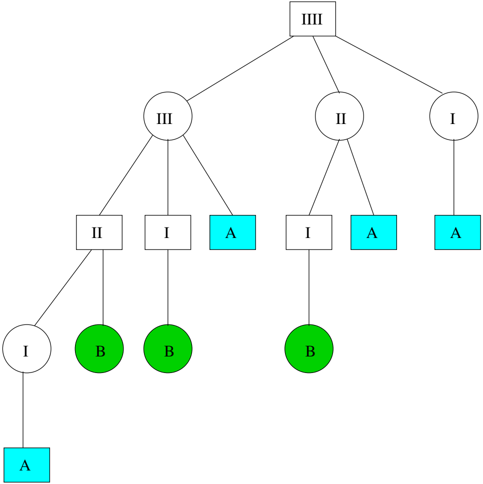
- Los juegos de antagonismo son un caso particular de problemas en los que la solución se busca en un árbol.
- En este caso, hay dos jugadores que se van turnando para realizar una jugada.
- Cada jugador tiene información completa del otro (es decir, no hay información oculta como en los juegos de cartas).
- Suelen ser juegos de suma nula: lo que gana un jugador lo pierde el otro.
- El objetivo de cada jugador consiste en maximizar su puntuación

Ejemplo: el juego del Nim

- Se dispone de N palillos en un montón.
- Dos jugadores se turnan para ir quitando uno, dos o tres palillos del montón.
- Pierde el último jugador que coge palillos del montón y lo deja vacío.
- Por ejemplo, inicialmente el montón tiene 4 palillos:
    - El jugador A toma un palillo: quedan 3 palillos en el montón.
    - El jugador B toma dos palillos: queda un solo palillo.
    - El jugador A se ve obligado a tomar el último palillo que queda, y pierde.
- ¿Podría haber ganado el jugador A?

Ejemplo: el juego del Nim

- Podemos ver las combinaciones de movimientos posibles mediante un árbol:



Ejemplo: el juego del Nim

- Cada nodo representa una situación del juego.
- Cada nivel representa las situaciones posibles para uno de los jugadores:
  - los nodos cuadrados son los movimientos del jugador A
  - los nodos circulares son los movimientos del jugador B
- Los nodos terminales representan los finales del juego y quién es el ganador.
- Explorando las distintas posibilidades, vemos que algunas pueden garantizar un ganador, o bien conseguir que tenga más posibilidades de ganar.
- Cada jugador debe hacer el movimiento que maximice sus posibilidades, independientemente de lo que haga el otro jugador.
- Para conocer las posibilidades de un movimiento es necesario recorrer el subárbol que está por debajo de él.

Esquemas algorítmicos. Juegos de Antagonismo

El procedimiento minimax

- Es un esquema genérico de búsqueda para juegos de antagonismo en las que los adversarios realizan movimientos alternativos.
- El contexto del problema se caracteriza por:
  a) En cada momento cada jugador conoce tanto sus posibilidades como las del adversario.
  b) Cuando uno de los adversarios tiene que efectuar una jugada, está interesado en saber como evolucionará el juego a partir de cada posible elección.
- En muchos casos, no es posible tener en cuenta todos los caminos que puede tomar el juego, debido a la explosión combinatoria de todos los posibles movimientos.

El procedimiento minimax (Cont.)

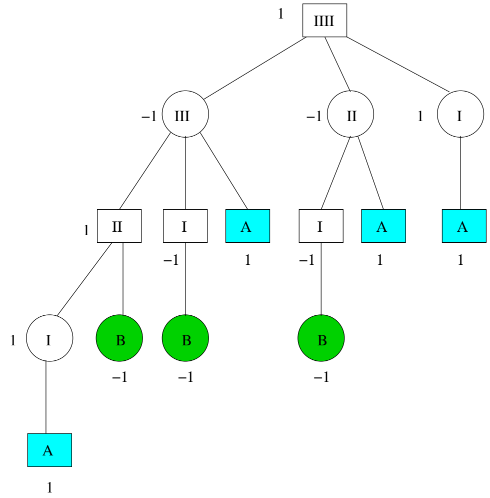
- Para resolver este tipo de problemas se intenta explorar de forma parcial el árbol de búsqueda:
  1) a partir de una situación determinada (un nodo determinado),
  2) con una profundidad limitada.
- El árbol de búsqueda se reconstruye cada vez que al jugador que representa la máquina le toca jugar. Después de cada jugada, se debe volver a construir el árbol para la nueva situación del juego.
- Por convención, las puntuaciones positivas hacen ganar a uno de los jugadores, denominado **MAX** o **maximizador**, mientras que las puntuaciones negativas hacen ganar al otro jugador, denominado **MIN** o **minimizador**,
- Cada nodo del árbol tiene asociado un valor de una función que determina la situación del juego en la configuración de dicho nodo.

El procedimiento minimax (Cont.)

- La función de evaluación es diferente para los nodos terminales respecto de los nodos intermedios.
- En el ejemplo del juego del Nim podemos utilizar las siguientes funciones:
  - Para los nodos terminales:
    $$V(X) = \begin{cases} 1 & \text{si en X es ganador el jugador MAX} \\ -1 & \text{si en X es ganador el jugador MIN} \end{cases}$$
  - Para los nodos intermedios:
    $$V(X) = \begin{cases} max\{V(hijos(X))\} & \text{si en X mueve el jugador MAX} \\ min\{V(hijos(X))\} & \text{si en X mueve el jugador MIN} \end{cases}$$
- Los nodos intermedios propagan la evaluación de los nodos terminales del subárbol al que pertenecen
- Cada nodo intermedio propaga hacia arriba el mejor valor de sus hijos.
- Si la raíz corresponde al jugador MAX, la mejor jugada corresponde a la rama que lleva desde la raíz al subárbol con el valor más alto de la función de evaluación.

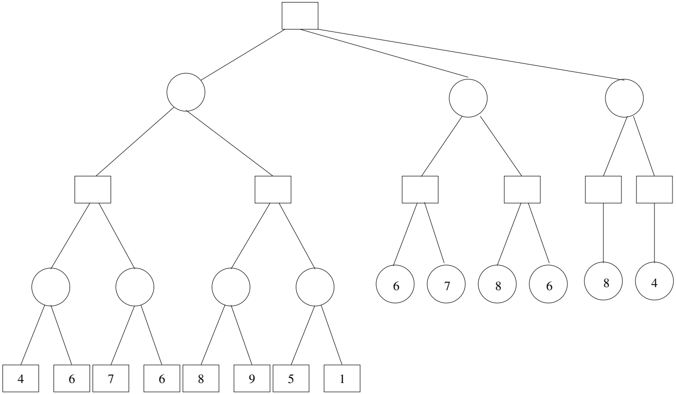
El procedimiento minimax (Cont.)

- El árbol de juego visto anteriormente queda etiquetado de la siguiente forma:



El procedimiento minimax (Cont.)

- Ejercicio: en el siguiente árbol, señala las estrategias ganadoras para el jugador MAX que realiza el primer movimiento.
- Un estado terminal es ganador para MAX si su valor es superior o igual a 6.



El procedimiento minimax (Cont.)

- En muchos casos el árbol completo a partir de una situación del juego hasta los nodos terminales es demasiado grande para tratarlo completamente (ajedrez, damas, etc.).
- Una posible solución consiste en generar el árbol parcial hasta una profundidad determinada, y en los nodos hoja utilizar una **función heurística** que aproxime el valor de la función de evaluación en esa situación del juego.
- La utilización de una heurística no garantiza el éxito: el camino seleccionado es una aproximación "razonable" hacia la victoria o el empate.
- Imita el comportamiento humano en este tipo de juegos.

Esquemas algorítmicos. Juegos de Antagonismo

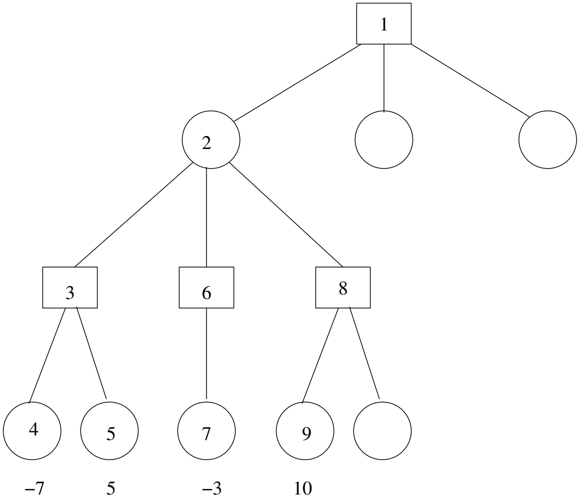
1. Características generales
2. Ejemplo: el juego del Nim
3. El procedimiento minimax
4. Poda alfa-beta
5. Las tres en raya

Poda alfa-beta

- Otra forma de mejorar el procedimiento minimax consiste en podar las ramas que no sean relevantes.
- En el procedimiento minimax, los nodos que no son relevantes son aquellos que no van a aportar información adicional: su evaluación no va a modificar el máximo o el mínimo que se está calculando.
- La poda alfa-beta no modifica el valor del máximo o mínimo calculado.
- Lo veremos con un ejemplo

Poda alfa-beta (Cont.)

- Supongamos un juego de dos jugadores con el siguiente árbol de búsqueda de soluciones:



Poda alfa-beta (Cont.)

- Los números dentro de los nodos representan el orden en el que se recorren (en profundidad).
- Las etiquetas debajo de las hojas son el resultado de la función de evaluación sobre ellas.
- El nodo 3 (correspondiente al jugador MAX) tendrá 5 como resultado de la función de evaluación (max{-7,5})
- El nodo 6 tendrá -3 como resultado de la función de evaluación.
- Con estos dos resultados, podemos garantizar que el nodo que está por encima de estos dos (el nodo 2) va a tener como resultado máximo -3, pues corresponde al jugador MIN (min{5,-3,?})
- Con este resultado, se evalúan los nodos 8 y 9. Con el resultado del nodo 9, podemos comprobar que el nodo 8 (MAX) va a tener como mínimo el valor 10 (max{10,?}).
- Como el nodo 2 no va a tener un valor superior a -3, y el nodo 8 no va a tener un valor inferior a 10, **es posible podar los demás nodos que están por debajo de 8**.

Poda alfa-beta (Cont.)

- α-valor:
  - Se denomina α-valor de una posición MAX a una **cota inferior** del valor que puede asignarse en esa posición.
  - Si el valor de una posición MIN es **menor o igual** que el α-valor de su padre $\rightarrow$ no se generan los hijos de esa posición.
- β-valor:
  - Se denomina β-valor de una posición MIN a una **cota superior** del valor que puede asignarse en esa posición.
  - Si el valor de una posición MAX es **mayor o igual** que el β-valor de su padre $\rightarrow$ no se generan los hijos de esa posición.
- Los valores α se inicializan a $-\infty$, y se van incrementando conforme se van explorando los subárboles de sus hijos
- Del mismo modo, los valores β se inicializan a ∞

Poda alfa-beta (Cont.)

	1 α	2 β	3 α	4	5	6 α	7	8 α	9
1	$-\infty$								
2	$-\infty$	∞							
3	$-\infty$	∞	$-\infty$						
4	$-\infty$	∞	$-\infty$	-7					
3	$-\infty$	∞	-7	-7					
5	$-\infty$	∞	-7	-7	5				
3	$-\infty$	∞	5	-7	5				
2	$-\infty$	5	5	-7	5				
6	$-\infty$	5	5	-7	5	$-\infty$			
7	$-\infty$	5	5	-7	5	$-\infty$	-3		
6	$-\infty$	5	5	-7	5	-3	-3		
2	$-\infty$	-3	5	-7	5	-3	-3		
8	$-\infty$	-3	5	-7	5	-3	-3	$-\infty$	
9	$-\infty$	-3	5	-7	5	-3	-3	$-\infty$	10
8	$-\infty$	-3	5	-7	5	-3	-3	10	10
2	$-\infty$	-3	5	-7	5	-3	-3	10	10
1	-3	-3	5	-7	5	-3	-3	10	10

Poda alfa-beta (Cont.)

```
fun A(X,prof,β)
  α ← −∞
  si terminal(X) ∨ prof = 0 entonces
    α ← valor(X) //valor o heurística
  si no
    i ← 1
    mientras existeHijo(X,i) ∧ α < β hacer
      v ← B(hijo(X,i),prof-1,α)
      si α < v entonces α ← v
      i ← i+1
    fin mientras
  fin si
  devolver α
fin fun
```

```
fun B(X,prof,α)
  β ← ∞
  si terminal(X) ∨ prof = 0 entonces
    β ← valor(X) //valor o heurística
  si no
    i ← 1
    mientras existeHijo(X,i) ∧ β > α hacer
      v ← A(hijo(X,i),prof-1,β)
      si β > v entonces β ← v
      i ← i+1
    fin mientras
  fin si
  devolver β
fin fun
```

Poda alfa-beta (Cont.)

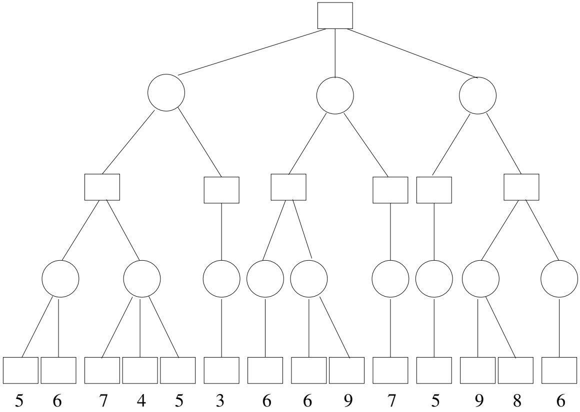
- La primera llamada debería ser:
  A(1,profMax,∞)
- Para obtener el movimiento a realizar, debería realizarse la primera llamada de la siguiente forma:

  $\alpha \leftarrow -\infty$
  mover $\leftarrow$ 0
  i $\leftarrow$ 1
  mientras existeHijo(X,i) **hacer**
      v $\leftarrow$ B(hijo(X,i),profMax-1,α)
      si $\alpha <$ v **entonces**
          $\alpha \leftarrow$ v
          mover $\leftarrow$ i
      fin si
      i $\leftarrow$ i+1
  fin mientras
- La variable mover contiene el movimiento a realizar

Poda alfa-beta (Cont.)

- Ejercicio: ¿Qué nodos no son evaluados mediante la poda alfa-beta en el siguiente árbol?

  (http://en.wikipedia.org/wiki/Alpha-beta_pruning)



Esquemas algorítmicos. Juegos de Antagonismo

1. Características generales
2. Ejemplo: el juego del Nim
3. El procedimiento minimax
4. La poda alfa-beta
5. $\boxed{\text{Las tres en raya}}$

Las tres en raya

- Este juego consiste en un tablero 3×3 inicialmente vacío.
- Los jugadores colocan alternativamente una de sus fichas en el tablero.
- Gana la partida el jugador que consigue situar tres de sus fichas en una misma línea: horizontal, vertical o diagonal.
- Consideramos la variante del juego que utiliza más de tres fichas para cada jugador, hasta llenar el tablero.

- Vamos a diseñar un algoritmo minimax con poda alfa-beta y heurística que decida el siguiente movimiento a realizar.

Las tres en raya (Cont.)

- El esquema visto anteriormente para el algoritmo minimax con poda alfa-beta puede utilizarse para este caso, aunque es necesario adaptarlo a las características del juego.

```
fun tictactoeAlfa(X[1..3,1..3],profundidad,β)
  α ← −∞ ; gana ← ganador(X)
  si gana = 'A' entonces α ← ∞
  si no si gana = 'B' ∨ tableroLleno(X) entonces
    si gana ≠ 'B' ∧ α < 0 entonces α ← 0
  si no si profundidad = 0 entonces
    α ← valoracion(X) // heurística
  si no
    desde i ← 1 hasta 3 hacer
      desde j ← 1 hasta 3 hacer
        si jugadaValida(X,i,j) entonces
          si ¬(α < β)entonces devolver α // poda
          v ← tictactoeBeta(aplicar(X,i,j,'A'),profundidad-1,α)
          si α < v entonces α ← v
        fin si
      fin desde
    fin desde
  fin si
  devolver α
fin fun
```

Las tres en raya (Cont.)

- La función que corresponde a los nodos MIN es la siguiente:

fun tictactoeBeta(X[1..3,1..3],profundidad,α)
  $\beta \leftarrow \infty$; gana $\leftarrow$ ganador(X)
  si gana = 'B' **entonces** $\beta \leftarrow -\infty$
  si no si gana = 'A' $\vee$ tableroLleno(X) **entonces**
    si gana $\neq$ 'A' $\wedge$ $\beta > 0$ **entonces** $\beta \leftarrow 0$
  si no si profundidad = 0 **entonces**
    $\beta \leftarrow$ valoracion(X) // heurística
  si no
    desde i $\leftarrow$ 1 **hasta** 3 **hacer**
      desde j $\leftarrow$ 1 **hasta** 3 **hacer**
        si jugadaValida(X,i,j) **entonces**
          si $\neg(\beta > \alpha)$**entonces devolver** β // poda
          v $\leftarrow$ tictactoeAlfa(aplicar(X,i,j,'B'),profundidad-1,β)
          si $\beta >$ v **entonces** $\beta \leftarrow$ v
        fin si
      fin desde
    fin desde
  fin si
  devolver β
fin fun

Las tres en raya (Cont.)

```
fun tableroLleno(X[1..3,1..3])
  // Devuelve cierto si el tablero esta lleno
  desde i ← 1 hasta 3 hacer
    desde j ← 1 hasta 3 hacer
      si vacio(X[i,j]) entonces devolver falso
    fin desde
  fin desde
  devolver cierto
fin fun

fun jugadaValida(X[1..3,1..3],i,j)
  // Determina si poner en (i,j) es correcto o no
  devolver vacio(X[i,j])
fin fun

fun aplicar(X[1..3,1..3],i,j,jug)
  // Devuelve la situacion del juego despues de aplicar el mov. (i,j) a X con jugador jug
  crear Y[1..3,1..3]
  Y ← X
  Y[i,j] ← jug
  devolver Y
fin fun
```

Las tres en raya (Cont.)

fun ganador(X[1..3,1..3])
  desde i $\leftarrow$ 1 **hasta** 3 **hacer**
    si $\neg$vacio(X[i,1]) $\wedge$ X[i,1] $=$ X[i,2] $\wedge$ X[i,2] $=$ X[i,3] **entonces devolver** X[i,1]
    si $\neg$vacio(X[1,i]) $\wedge$ X[1,i] $=$ X[2,i] $\wedge$ X[2,i] $=$ X[3,i] **entonces devolver** X[1,i]
  fin desde
  si $\neg$vacio(X[1,1]) $\wedge$ X[1,1] $=$ X[2,2] $\wedge$ X[2,2] $=$ X[3,3] **entonces devolver** X[1,1]
  si $\neg$vacio(X[1,3]) $\wedge$ X[1,3] $=$ X[2,2] $\wedge$ X[2,2] $=$ X[3,1] **entonces devolver** X[3,1]
  devolver ninguno
fin fun

Las tres en raya (Cont.)

```
fun valoracion(X[1..3,1..3]) //heuristica: h(X)=lineas posibles para A - lineas posibles para B
  crear filA[1..3], filB[1..3], colA[1..3], colB[1..3], diagA[1..2], diagB[1..2]
  desde i ← 1 hasta 3 hacer
    desde j ← 1 hasta 3 hacer
      si X[i,j] = 'A' entonces filA[i] ← filA[i]+1 ; colA[j] ← colA[j]+1
      si X[i,j] = 'B' entonces filB[i] ← filB[i]+1 ; colB[j] ← colB[j]+1
    fin desde
    si X[i,i] = 'A' entonces diagA[1] ← diagA[1]+1
    si X[i,i] = 'B' entonces diagB[1] ← diagB[1]+1
    si X[i,3-i+1] = 'A' entonces diagA[2] ← diagA[2]+1
    si X[i,3-i+1] = 'B' entonces diagB[2] ← diagB[2]+1
  fin desde
  desde i ← 1 hasta 3 hacer
    si filA[i] ≥ 0 ∧ filB[i] = 0 entonces promA ← promA + 1
    si filB[i] ≥ 0 ∧ filA[i] = 0 entonces promB ← promB + 1
    si colA[i] ≥ 0 ∧ colB[i] = 0 entonces promA ← promA + 1
    si colB[i] ≥ 0 ∧ colA[i] = 0 entonces promB ← promB + 1
  fin desde
  desde i ← 1 hasta 2 hacer
    si diagA[i] ≥ 0 ∧ diagB[i] = 0 entonces promA ← promA + 1
    si diagB[i] ≥ 0 ∧ diagA[i] = 0 entonces promB ← promB + 1
  fin desde
  devolver promA - promB
fin fun
```

Las tres en raya (Cont.)

- Para obtener el movimiento a realizar debe utilizarse el siguiente código:

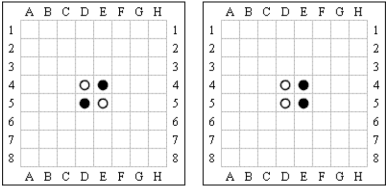
proc llamadorTictactoe(X[1..3,1..3],profMax,movI,movJ)
  $\alpha \leftarrow -\infty$
  movI $\leftarrow$ 1 ; movJ $\leftarrow$ 1
  desde i $\leftarrow$ 1 **hasta** 3 **hacer**
    desde j $\leftarrow$ 1 **hasta** 3 **hacer**
      si jugadaValida(X,i,j) **entonces**
        v $\leftarrow$ tictactoeBeta(aplicar(X,i,j,'A'),profMax-1,α)
        si $\alpha < v$ **entonces** $\alpha \leftarrow$ v ; movI $\leftarrow$ i ; movJ $\leftarrow$ j
      fin si
    fin desde
  fin desde
fin proc

Ejercicio: Reversi

- Debe modificarse el algoritmo anterior para que encuentre el siguiente movimiento a realizar en el juego **Reversi** (es más conocida la variante denominada **Otelo**©).
  - El tablero es en este caso $N \times N$.
  - Hay tantas piezas como casillas en el tablero, con un color distinto en cada cara (cada color corresponde a un jugador).
  - Cada jugador tiene la mitad de las piezas ($\neq$ Otelo).
  - En cada turno un jugador coloca una pieza con su color a la vista en una casilla libre, y voltea todas las piezas del color contrario que estén flanqueadas en línea recta por la pieza recién colocada y otra pieza del mismo color.
  - Si no se puede voltear ninguna pieza, el jugador que tiene el turno tiene que pasárselo al otro jugador.
  - Gana el juego el jugador que tiene más piezas de su color cuando ninguno de los jugadores puede colocar ninguna pieza más.

Ejercicio: Reversi (cont.)

- Situaciones iniciales del juego del Reversi (en el Otelo solamente se permite la situación de la izquierda):



- Ejemplo de movimiento del jugador blanco: