Costruzione del caso peggiore per un algoritmo deterministico

Pezzano Enrico

Un albero di un gioco $T_{3,6}$ è un albero uniforme avente tutti i nodi interni con 3 figli che corrispondono alle mosse eseguibili dalla configurazione del padre.

L'albero di sopra avrà parecchie foglie ($3^{2*6} = 9^6 = 531.441$ per essere precisi), quindi richiederebbe decisamente troppo tempo per essere disegnato a mano.
Incontriamo un problema nella valutazione del gioco quando vogliamo determinare il valore che restituisce la radice, tenendo conto che in ogni foglia vi è associato un numero reale corrispondente al valore del gioco e che ogni nodo, se si trova a distanza dispari dalla radice sarà di tipo MAX e restituirà il valore massimo dei suoi *d* figli; altrimenti sarà di tipo *MIN* e restituirà il valore minimo. Nel caso in cui ogni foglia potesse assumere solo il valore *0* o *1*, il funzionamento di *MAX* potrà essere considerato uguale a quello di ***OR*** e quello di *MIN* equivalente ad un ***AND***.
Di seguito un **esempio** di albero per cui un algoritmo deterministico dovrebbe valutare tutte le foglie:
considerando una procedura deterministica ricorsiva nella quale l'albero viene letto da sinistra verso destra, partiamo ricorsivamente e scendiamo nei sotto-alberi a sinistra fino a raggiungere una foglia. A questo punto osserviamo il padre della foglia in questione:

- **caso OR**

  per raggiungere il nostro obiettivo bisogna fare in modo che non si possa stabilire il valore del padre senza aver prima analizzato gli altri due figli; sarà quindi necessario che il figlio centrale e quello di sinistra siano *0* e di conseguenza il figlio restante dovrà essere *1*. Il figlio di destra dovrà avere obbligatoriamente il valore *1*, facendo in modo che anche il padre assuma il suo valore (*1*), così che l'algoritmo sia costretto a leggere gli altri sotto-alberi per dare un valore all'*AND* successivo. In questo modo l'algoritmo analizzerà il sotto-albero subito alla destra di quello appena studiato e sarà necessario che anche quest'ultimo sia *1* (e siccome è anche lui un *OR*, dovrà avere le caratteristiche del sotto-albero visto in precedenza).
  Una volta osservato anche il sotto-albero (il cui valore sarà *1*) l'algoritmo non potrà ancora dare un valore all'*AND*, perchè dipenderà dal valore del terzo ed ultimo sotto-albero, il quale ci interessa abbia valore *0*, in modo da rendere l'*AND* nullo e costringere l'algoritmo a dover analizzare i restanti sotto-alberi per dare un valore all'*OR* successivo, analogamente al primo *OR* studiato, e così via fino alla radice.

- **caso AND**

  diversamente dal *caso OR*, bisogna "gestire" l'algoritmo negandogli la possibilità di escludere se l'*AND* in questione assuma valore *0* o *1*, senza avere analizzato prima gli altri due figli; questi ultimi dovranno essere *1* ed il restante *0*, rendendo *0* il valore del padre ed obbligando l'algoritmo a dover analizzare i restanti due sotto-alberi per poter assegnare un valore al seguente *OR*.
  Studiando il secondo sotto-albero, per dare un valore all'*OR* sarà necessario che anche lui sia *0* e quindi dovrà essere identico al sotto-albero a sinistra. Così facendo, l'algoritmo non sarà in grado di assegnare un valore all'*OR* fino al momento in cui non avrà analizzato anche il restante sotto-albero, che dovrà diventare *1* (facendo sì che l'*AND* successivo dipenda dal suo ultimo figlio e così via fino alla radice).

Ricapitolando: bisogna obbligare l'algoritmo ad analizzare tutto l'albero; perciò è necessario che, studiando un *OR*, i due figli più a sinistra abbiano valore *0*, rendendo l'*OR* dipendente dall'ultimo figlio (quello di destra), che sarà *0* o *1* a seconda che sia uno dei due figli più a sinistra o del figlio più a destra di un *AND*.
Analizzando un *AND*, è necessario che l'algoritmo non possa escludere (senza prima aver osservato tutti e tre i figli) che possa assumere valore *1*; quindi i due figli più a sinistra saranno *1* e l'ultimo, a seconda se ci troviamo nei due figli più a sinistra o in quello più a destra di un *OR*, sarà rispettivamente *0* o *1*.