Modulo A - Laboratorio

Algoritmi di Ricerca per Giochi

Giochi come tris, dama, scacchi, go sono

- deterministici (non ci sono elementi di casualità)
- a due giocatori
- a turni
- a informazione perfetta (lo stato del gioco è noto)
- a somma zero (una mossa che avvantaggia un giocatore svantaggia l'altro)

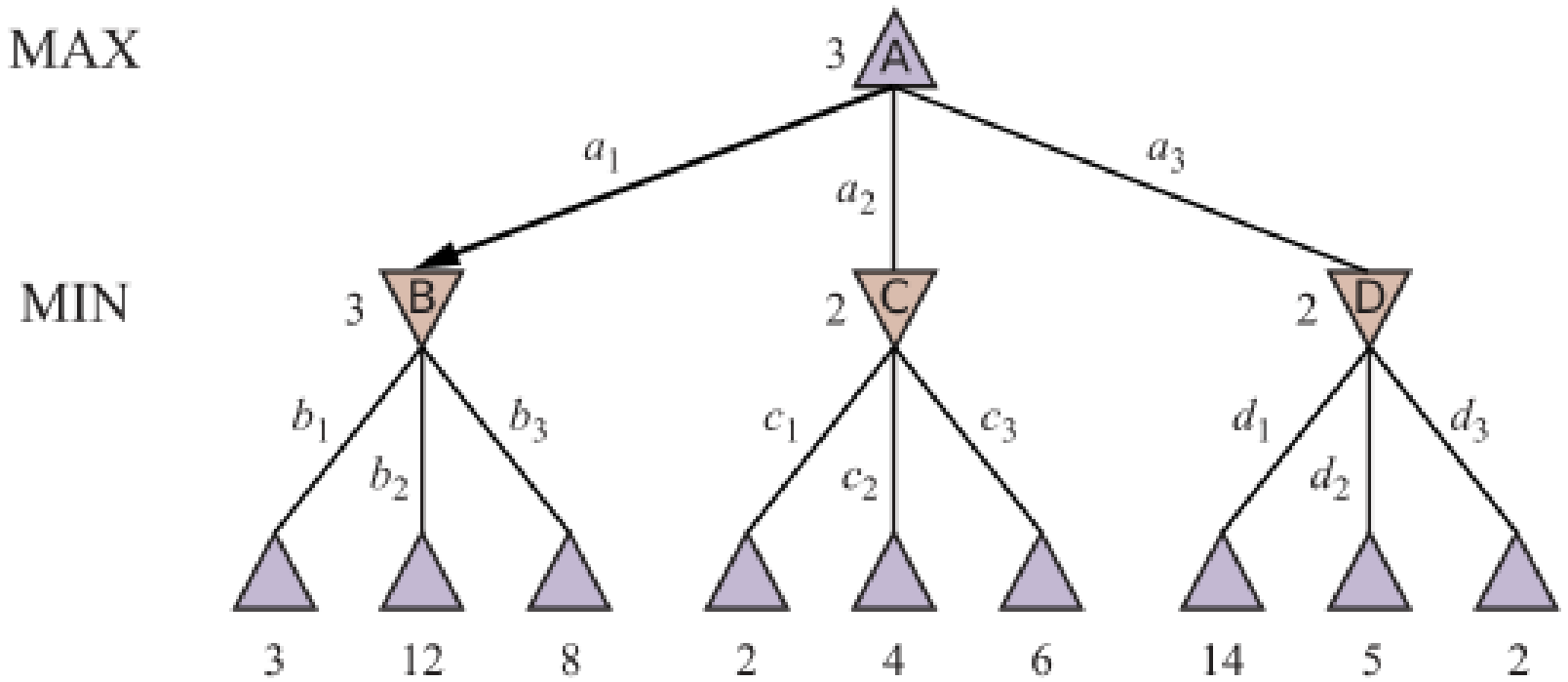
Caratterizzazione di Giochi

- `S0`: lo stato iniziale del gioco
- `To-Move(s)`: chi deve muovere nello stato `s`
- `Actions(s)`: l'insieme di azioni possibili nello stato `s`
- `Result(s,a)`: definisce lo stato risultante dall'applicazione dell'azione `a` nello stato `s`
- `Is-Terminal(s)`: controlla se lo stato `s` è terminale
- `Utility(s,p)`: definisce l'utilità dello stato terminale `s` per il giocatore `p`

Come Selezionare la Mossa Migliore?

- Il giocatore muoverà cercando di massimizzare la sua utilità. Chiamiamo questo giocatore MAX
- L'avversario cercherà di minimizzare l'utilità di MAX; lo chiamiamo MIN
- A una mossa di MAX, MIN risponderà con la sua mossa che minimizza l'utilità di MAX
- Quindi MAX deve selezionare la mossa per cui è massimo il minimo delle utilità delle risposte di MIN

Esempio



min-max

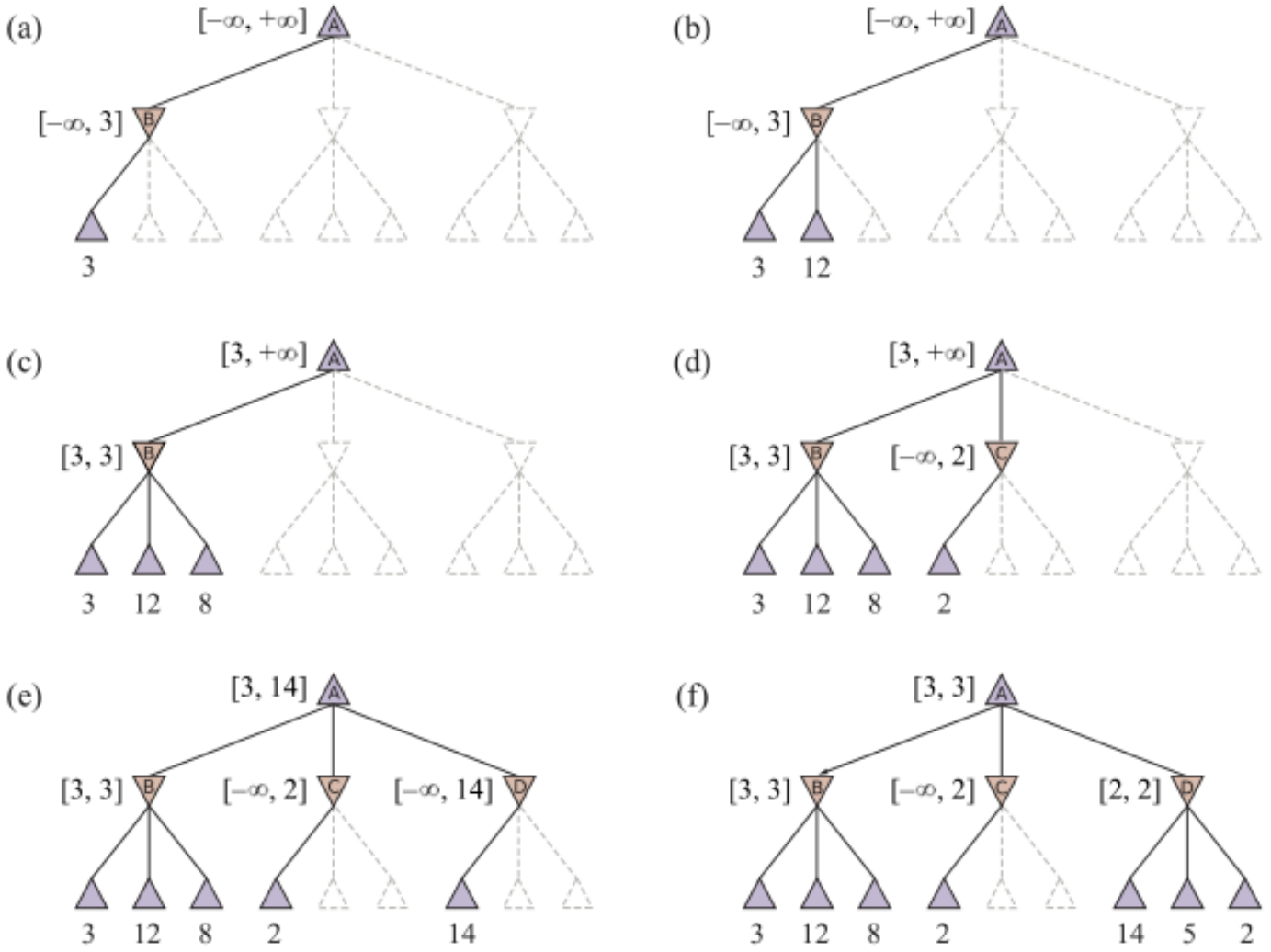
```
function MINIMAX-SEARCH(game, state) returns an action
    player ← game.TO-MOVE(state)
    value, move ← MAX-VALUE(game, state)
    return move

function MAX-VALUE(game, state) returns a (utility, move) pair
    if game.IS-TERMINAL(state) then return game.UTILITY(state, player), null
    v ← −∞
    for each a in game.ACTIONS(state) do
        v2, a2 ← MIN-VALUE(game, game.RESULT(state, a))
        if v2 > v then
            v, move ← v2, a
    return v, move

function MIN-VALUE(game, state) returns a (utility, move) pair
    if game.IS-TERMINAL(state) then return game.UTILITY(state, player), null
    v ← +∞
    for each a in game.ACTIONS(state) do
        v2, a2 ← MAX-VALUE(game, game.RESULT(state, a))
        if v2 < v then
            v, move ← v2, a
    return v, move
```

Tagli alfa-beta

- In certi casi, si può dire con certezza che un nodo avrà un valore peggiore, per il giocatore che lo sta considerando, del migliore dei suoi fratelli
- In questi casi si può smettere di espandere il nodo (‘tagliare’ i rami rimanenti dell’albero di cui è radice)



alfa-beta

```
function ALPHA-BETA-SEARCH(game, state) returns an action
  player ← game.TO-MOVE(state)
  value, move ← MAX-VALUE(game, state, −∞, +∞)
  return move

function MAX-VALUE(game, state, α, β) returns a (utility, move) pair
  if game.IS-TERMINAL(state) then return game.UTILITY(state, player), null
  v ← −∞
  for each a in game.ACTIONS(state) do
    v2, a2 ← MIN-VALUE(game, game.RESULT(state, a), α, β)
    if v2 > v then
      v, move ← v2, a
      α ← MAX(α, v)
    if v ≥ β then return v, move
  return v, move

function MIN-VALUE(game, state, α, β) returns a (utility, move) pair
  if game.IS-TERMINAL(state) then return game.UTILITY(state, player), null
  v ← +∞
  for each a in game.ACTIONS(state) do
    v2, a2 ← MAX-VALUE(game, game.RESULT(state, a), α, β)
    if v2 < v then
      v, move ← v2, a
      β ← MIN(β, v)
    if v ≤ α then return v, move
  return v, move
```

Implementazione di un Motore per il Gioco della Dama

1. Implementare l’algoritmo min-max
2. Implementare l’algoritmo alfa-beta
3. Implementare alternative per il calcolo dell’utilità