Tìm kiếm đối kháng - trò chơi

Trí tuệ nhân tạo

HK1, 2022 - 2023

Nội dung

Trò chơi vs. Vấn đề tìm kiếm

- Đối thủ không thể dự đoán
- Môi trường cạnh tranh: mục đích của các tác nhân xung đột với nhau
- Giới hạn thời gian
- Ví dụ về độ phức tạp:
  - Cờ: b = 35, d = 100 $\rightarrow$ Kích thước cây: $\approx 10^{154}$
  - : Go: b = 1000

Các loại games







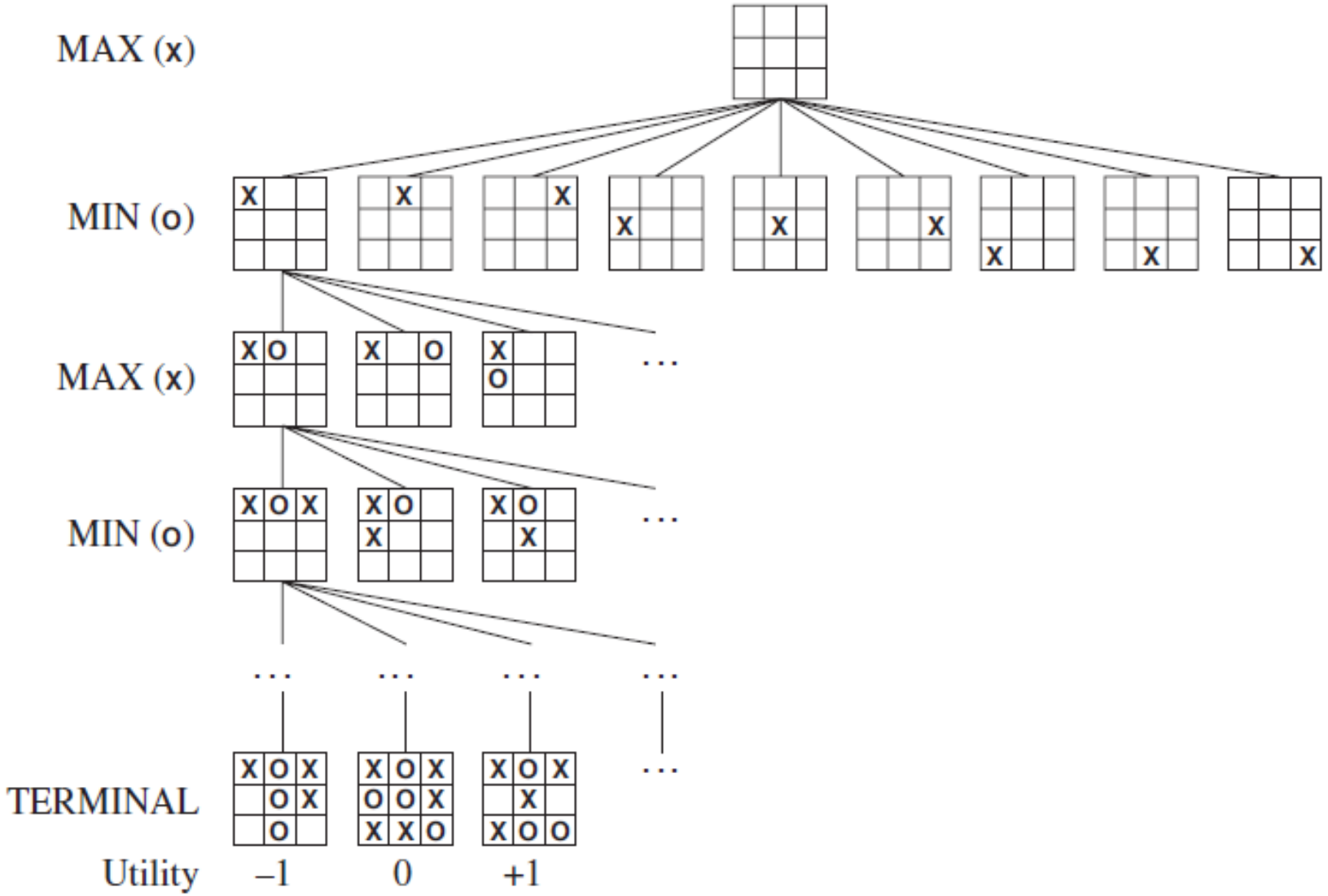
Các giả định

- Giả định chỉ có 02 người chơi
- Không có yếu tố may rủi
- Cả hai người chơi đều có tri thức hoàn chỉnh về trạng thái của trò chơi
- Trò chơi có tổng bằng 0 (zero-sum games)
  - là tình huống mà người chơi cạnh tranh nhau để hưởng số phần thưởng cố định $\rightarrow$ người này được thì người kia mất
  - Mỗi người chơi thắng ($+1$), thua (0), hòa ($1/2$)
- Người chơi hợp lý (Rational Players)

Thiết lập bài toán

- Hai người chơi: MAX và MIN
- MAX: đi trước, sau đó lần lượt các đấu thủ đến khi game kết thúc
  - Người thắng được thưởng, người thua bị phạt
- Trò chơi định nghĩa như bài toán tìm kiếm
  - S_0 - Trạng thái bắt đầu: xác định trò chơi khi bắt đầu được thiết lập như thế nào
  - PLAYER(s): MAX hay MIN sẽ chơi
  - ACTIONS(s) - Hàm chuyển trạng thái: danh sách các bước đi hợp lệ
  - RESULT(s, a) - Mô hình chuyển đổi: kết quả của việc thực hiện bước a khi đang ở trạng thái s
  - TERMINAL-TEST(s): kiểm tra game đã kết thúc?
  - UTILITY(s, p) - Hàm tiện ích: giá trị (số) của trạng thái s đối với người chơi p

Cây trò chơi Tic-Tac-Toe



Cờ (Chess)

- Độ phức tạp
  - $b \approx 35$
  - $d \approx 100$
  - Cây tìm kiếm $\approx 10^{154}$ nút

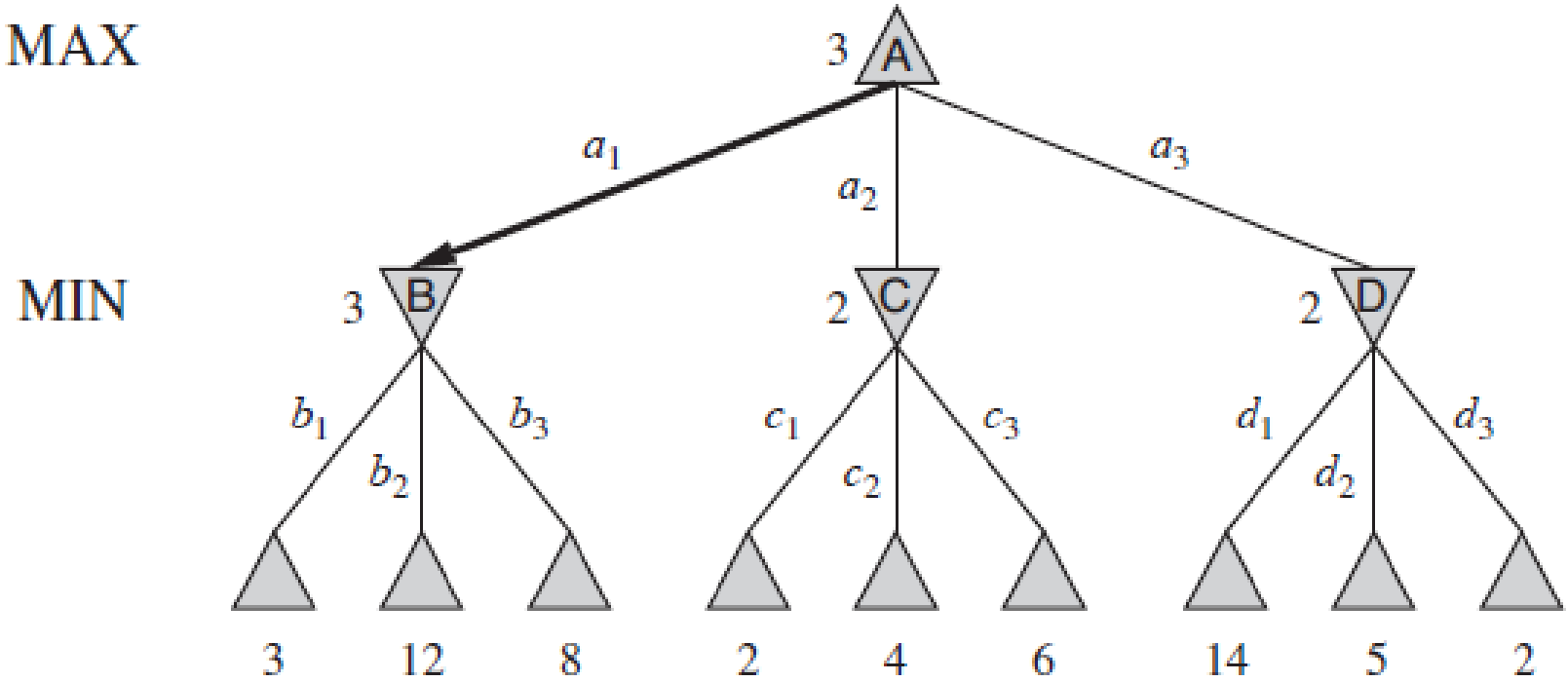
  $\rightarrow$ Không thể thực hiện bằng cách tìm kiếm
- Deep Blue
- Hiện tại và tương lai, dạy cho máy tính **học** cách chơi

Quyết định tối ưu (Optimal Decisions)

- Trong vấn đề tìm kiếm thông thường:
  - Giải pháp tối ưu: chuỗi hành động dẫn đến trạng thái đích
- Trong bài toán trò chơi:
  - Đường tìm kiếm bảo đảm một người chơi thắng
  - Chiến lược tối ưu có thể xác định từ giá trị **minimax** của mỗi nút

$$\text{MINIMAX}(s) = \begin{cases} \text{UTILITY}(s) & \text{nếu TERMINAL-TEST}(s) \\ \max_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{nếu PLAYER}(s) = \text{MAX} \\ \min_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{nếu PLAYER}(s) = \text{MIN} \end{cases}$$

Cây trò chơi Tic-Tac-Toe



Giải thuật Minimax

- John von Neumann đưa ra kỹ thuật tìm kiếm gọi là Minimax
- Người chơi MAX đấu với một đối thủ
  - Mục tiêu ngược lại với đối thủ
  - MAX cố gắng *tối đa hóa* nước đi của mình và giảm thiểu kết quả của nước đi đối thử (MIN)
- Hàm tiện ích (Utility function): sẽ cho biết vị trí của người chơi là tốt/xấu.

Giải thuật Minimax

- Định nghĩa cách chơi tối ưu cho MAX giả định MIN chơi một cách tối ưu:
  - Tối đa hóa kết quả đầu ra của trường hợp xấu nhất đối với MAX
- Minimax sử dụng DFS để duyệt qua cây trò chơi
- Áp dụng hàm tiện ích cho các trạng thái cuối

Giải thuật Minimax

function MINIMAX-DECISION(*state*) **returns** *an action*
  return $\arg\max_{a \in \text{ACTIONS}(s)}$ MIN-VALUE(RESULT(*state*, *a*))

---

function MAX-VALUE(*state*) **returns** *a utility value*
  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  $v \leftarrow -\infty$
  for each *a* **in** ACTIONS(*state*) **do**
    $v \leftarrow$ MAX(v, MIN-VALUE(RESULT(s, a)))
  return v

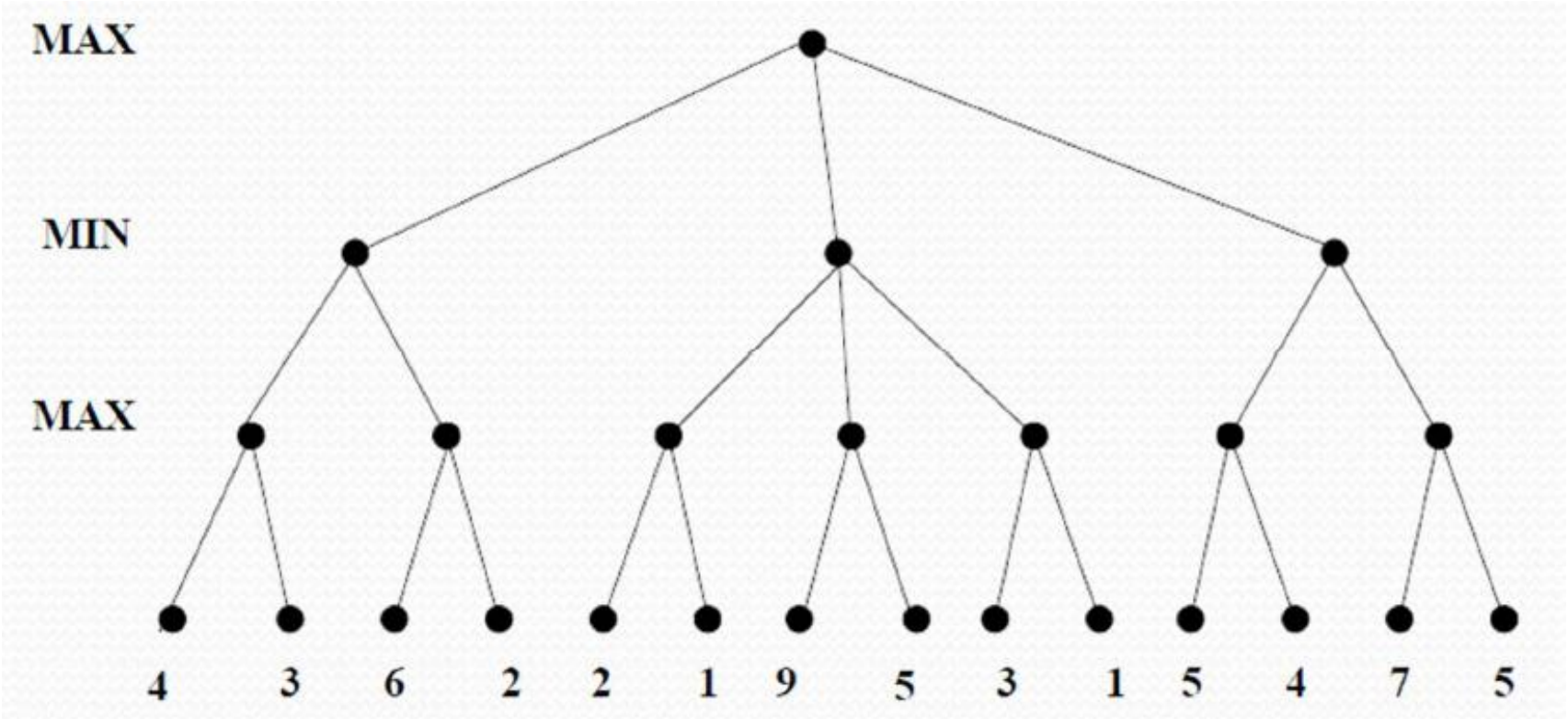
---

function MIN-VALUE(*state*) **returns** *a utility value*
  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  $v \leftarrow \infty$
  for each *a* **in** ACTIONS(*state*) **do**
    $v \leftarrow$ MIN(v, MAX-VALUE(RESULT(s, a)))
  return v

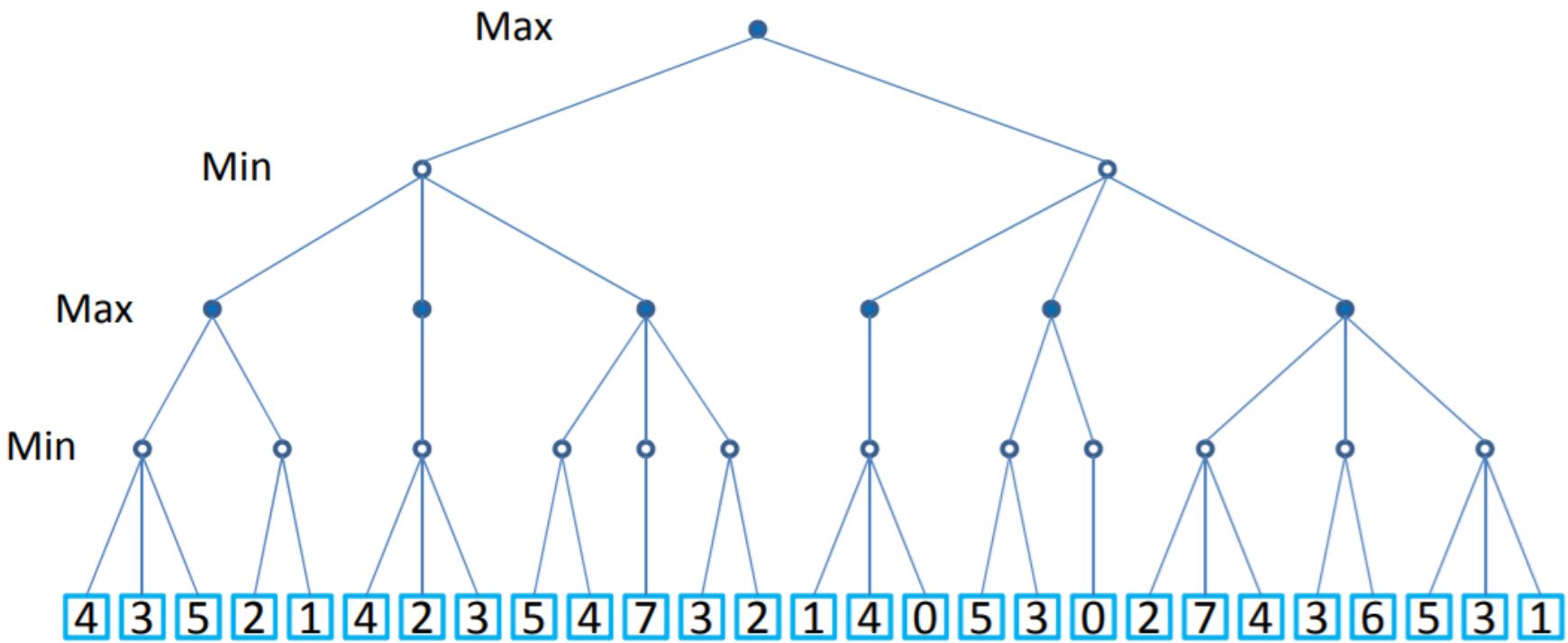
Giải thuật Minimax

- Đầy đủ:
  - Có (nếu cây hữu hạn)
- Tối ưu:
  - Có
- Độ phức tạp thời gian:
  - $O(b^m)$
- Độ phức tạp không gian:
  - $O(bm)$

Ví dụ:



Ví dụ:



Vấn đề với tìm kiếm Minimax

- Số trạng thái game là cấp số nhân với số lần di chuyển
  - Không duyệt tất cả các nút
  - Tỉa (pruning): bỏ những nhánh không ảnh hưởng đến quyết định cuối cùng
- Xem xét trước trong giới hạn
  - Giới hạn độ sâu cho các lần tìm kiếm
  - Giống như cách chơi cờ: xem xét trước một vài nước đi và xem đường nào tốt nhất

Cắt tỉa $\alpha - \beta$

Ý tưởng:

- Nếu một nhánh tìm kiếm nào đó không thể cải thiện giá trị hàm tiện ích mà MAX (hoặc MIN) đã có thì MAX (hoặc MIN) không cần xét đến nhánh tìm kiếm đó nữa
- α: giá trị tùy chọn tốt nhất (cao nhất) được tìm thấy dọc theo đường đang xem xét cho MAX
- β: giá trị tùy chọn tốt nhất (thấp nhất) được tìm thấy dọc theo đường đang xem xét cho MIN

Cắt tỉa $\alpha - \beta$

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

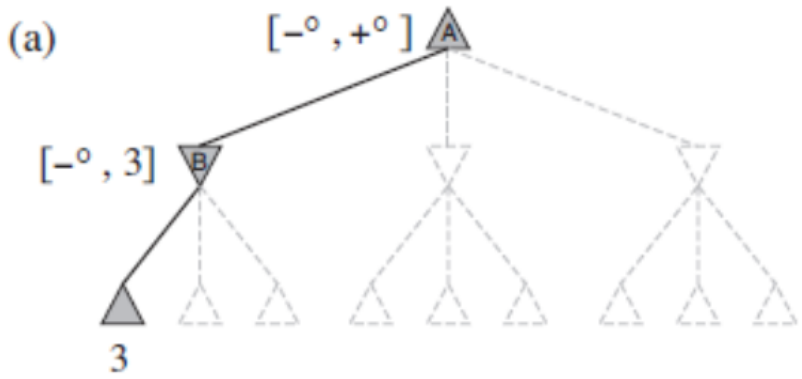
---

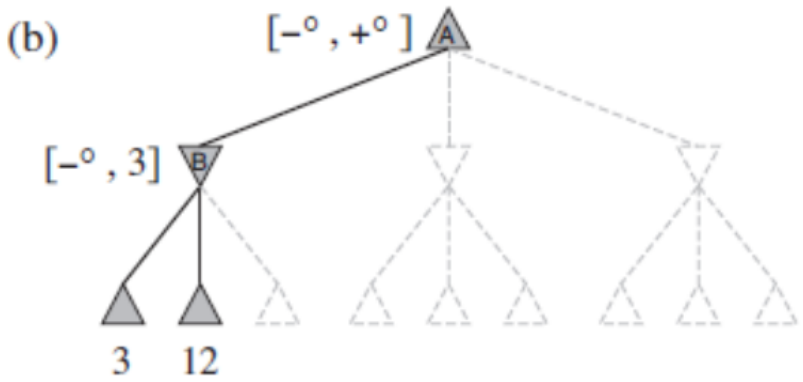
function MAX-VALUE($state, \alpha, \beta$) **returns** *a utility value*
  if TERMINAL-TEST($state$) **then return** UTILITY($state$)
  $v \leftarrow -\infty$
  for each a **in** ACTIONS($state$) **do**
    $v \leftarrow$ MAX(v, MIN-VALUE(RESULT(s,a), α, β))
    if $v \geq \beta$ **then return** v
    $\alpha \leftarrow$ MAX(α, v)
  return v

---

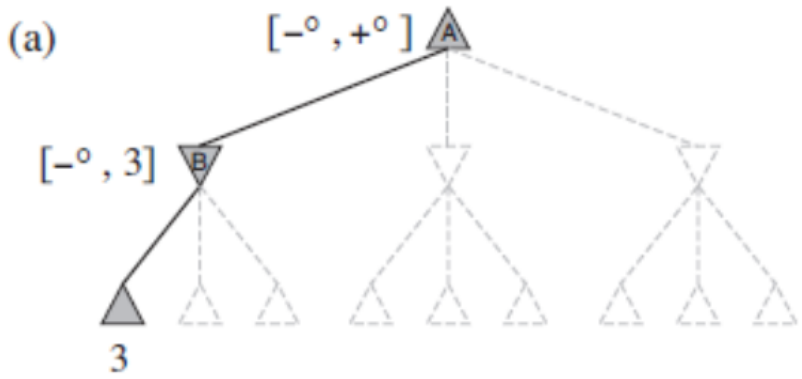
function MIN-VALUE($state, \alpha, \beta$) **returns** *a utility value*
  if TERMINAL-TEST($state$) **then return** UTILITY($state$)
  $v \leftarrow +\infty$
  for each a **in** ACTIONS($state$) **do**
    $v \leftarrow$ MIN(v, MAX-VALUE(RESULT(s,a) , α, β))
    if $v \leq \alpha$ **then return** v
    $\beta \leftarrow$ MIN(β, v)
  return v

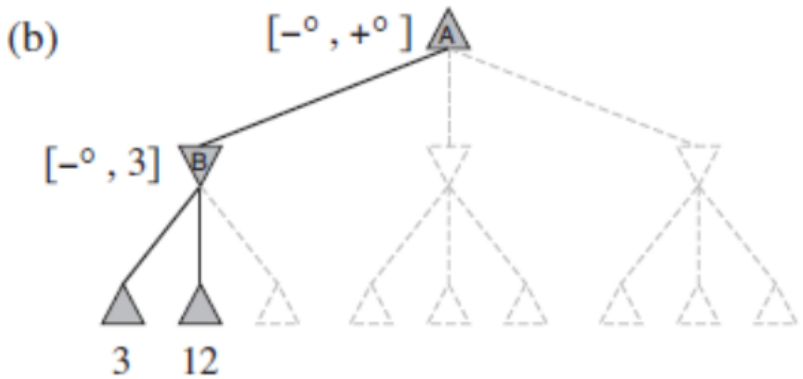
Cắt tỉa $\alpha - \beta$, Ví dụ

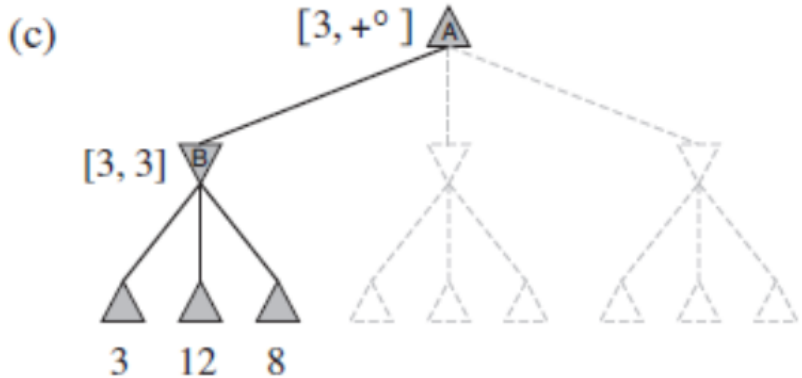


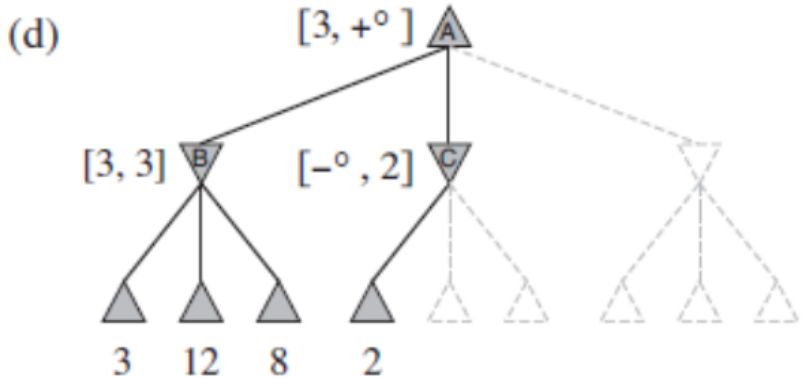


Cắt tỉa $\alpha - \beta$, Ví dụ

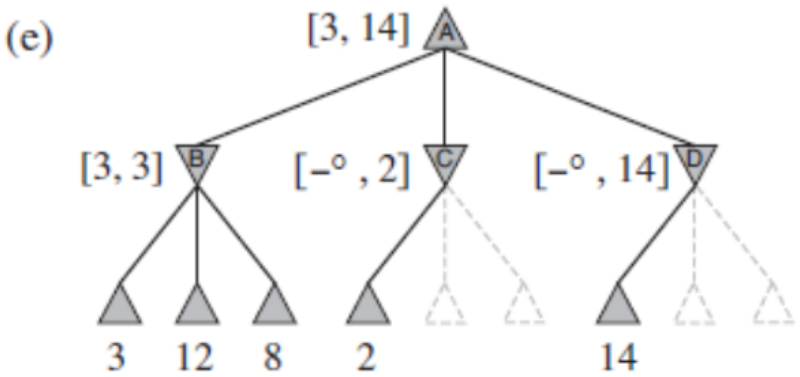


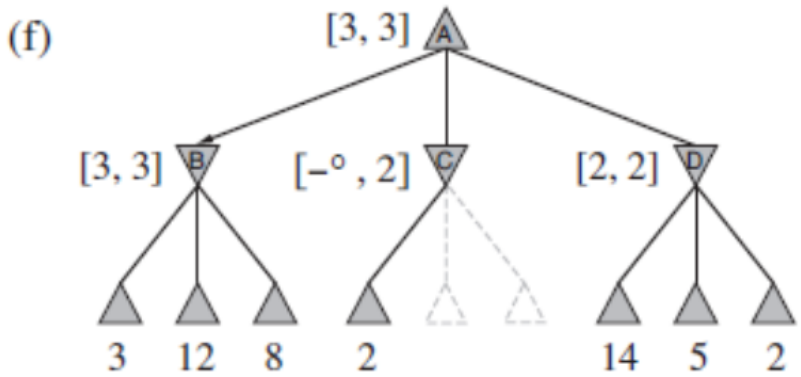






Cắt tỉa $\alpha - \beta$, Ví dụ





Cắt tỉa $\alpha - \beta$

- Cắt tỉa *không ảnh hưởng kết quả cuối cùng*
  - Tốt nhất: giảm kích thước cây
  - Xấu nhất: như giải thuật Minimax
- Thứ tự di chuyển tốt cải thiện hiệu quả của việc cắt tỉa
- Phức tạp thời gian: $O(b^{d/2})$