

# 博弈树搜索

---

# 目录

## 1. 理论课内容回顾

1.1 两玩家零和博弈问题

1.2 博弈树

1.3 Minimax搜索

1.4 Alpha-beta剪枝

# 1 两玩家零和博弈问题

## □ 两名玩家轮流行动，进行博弈

- 有限：行动的个数有限
- 确定性：不存在随机性
- 信息完备性：博弈双方知道所处状态的全部信息
- 零和性：一方的损失相当于另一方的收益，总收益为0

□ 结局有三种可能：玩家A获胜、玩家B获胜、平局（或两种可能，无平局）

# An example: Rock, Paper, Scissors

- ❑ Scissors cut paper, paper covers rock, rock smashes scissors
- ❑ Represented as a matrix: Player I chooses a row, Player II chooses a column
- ❑ Payoff to each player in each cell (Pl.I / Pl.II)
- ❑ 1: win, 0: tie, -1: loss
- ❑ so it's zero-sum

		Player II		
		R	P	S
Player I	R	0/0	-1/1	1/-1
	P	1/-1	0/0	-1/1
	S	-1/1	1/-1	0/0

# 1.2 博弈树

- 节点（node）：表示问题的状态（state）。
  - 分为内部节点（interior node）和叶子节点（leaf node）
- 扩展节点：行动（action）。
- 双方轮流扩展节点：两个玩家的行动逐层交替出现。
- 博弈树的值（game tree value）：博弈树搜索的目的，找出对双方都是最优的子节点的值。给定叶子节点的效益值，搜索内部节点的值。
- 评价函数（evaluator）：对当前节点的优劣评分。在有深度限制时，原来的内部节点会充当叶子节点的作用，此时以评价函数值作为效益值的估计。

## 1.2 博弈树

□ 博弈问题对人的深层次的知识研究提出了严峻的挑战。如何表示博弈问题的状态，博弈过程和博弈取胜的知识，这是目前人类仍在探讨之中的问题。要提高博弈问题求解程序的效率，应作到如下两点：

- 改进生成过程，使之只生成好的走步，如按棋谱的方法生成下一步；
- 改进测试过程，使最好的步骤能够及时被确认。

## 1.2 博弈树

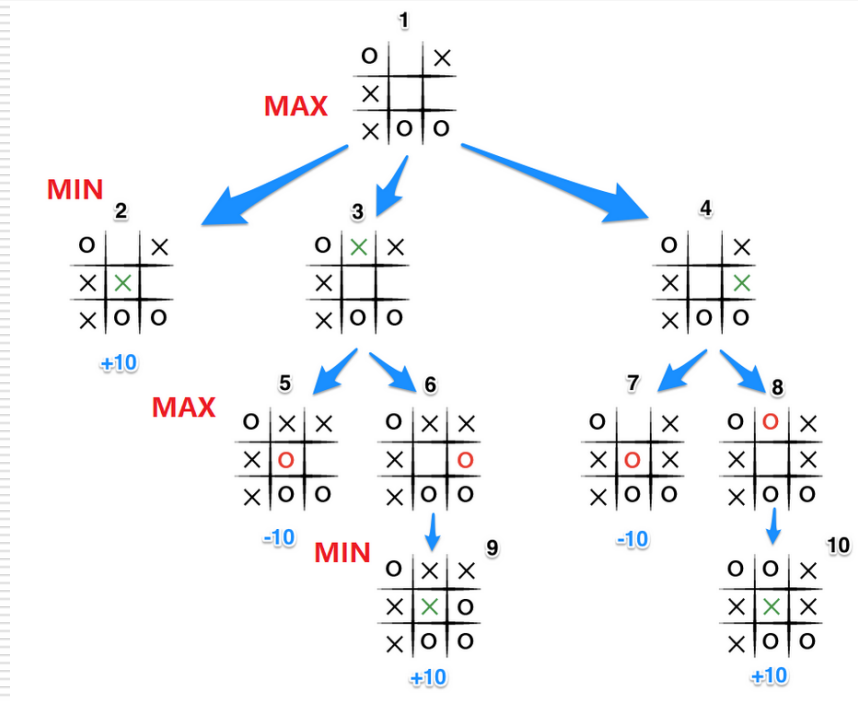
- 要达到上述目的有效途径是使用启发式方法引导搜索过程，使其只生成可能赢的走步。而这样的博弈程序应具备：
  - 一个好的（即只产生可能赢棋步骤的）生成过程。
  - 一个好的静态估计函数。
- 下面介绍博弈中两种最基本的搜索方法。

# 1.3 Minimax搜索

## □ 假设：

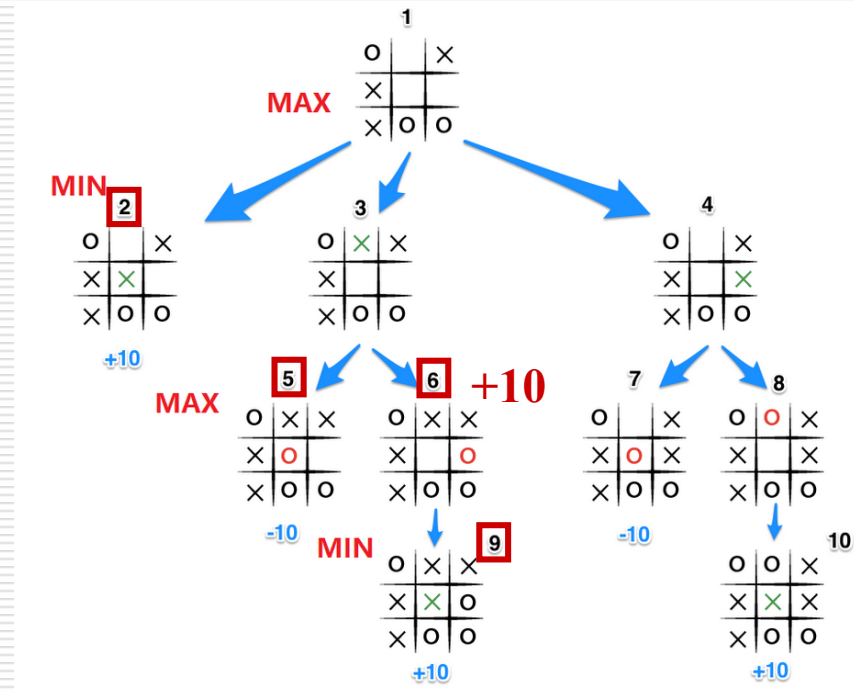
- 玩家A和玩家B的行动逐层交替；
- A和B的利益关系对立，即假设A要使分数更大，B就要使分数更小；
- A和B均采用最优策略。

□ Minimax搜索：找到博弈树中内部节点的值，其中Max节点（A）的每一步扩展要使收益最大，Min节点（B）的扩展要使收益最小。

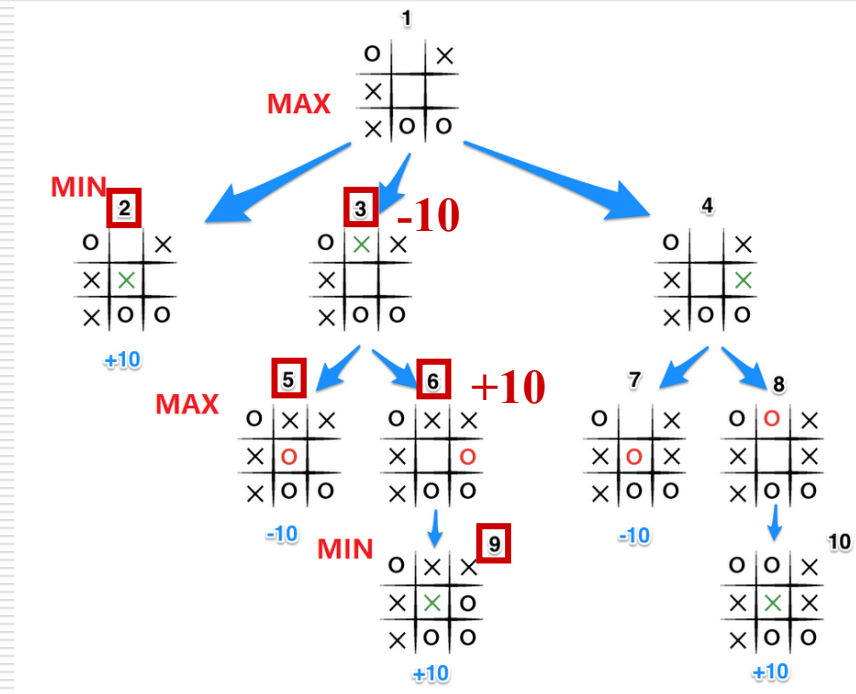




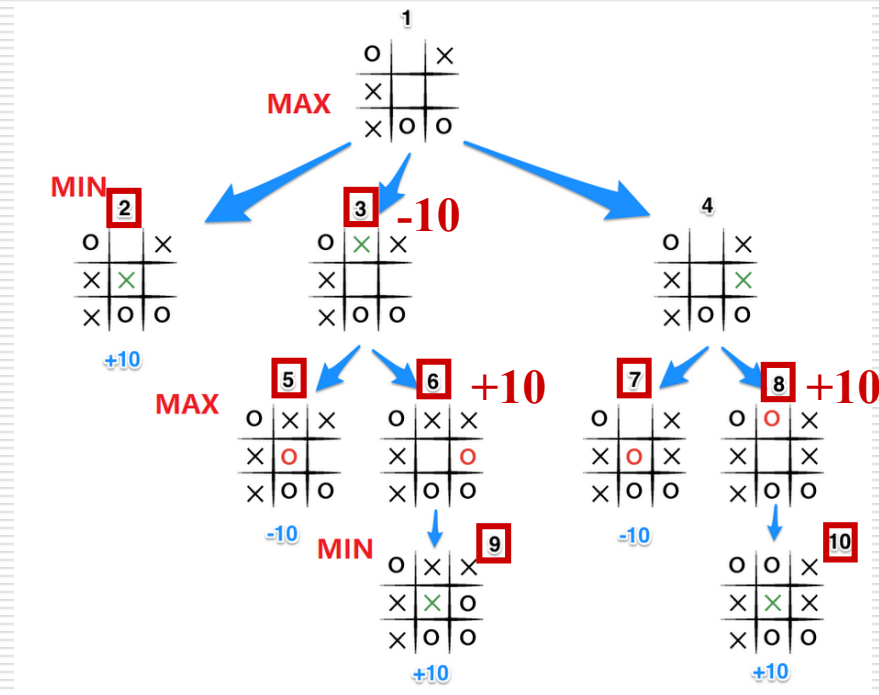
# 1.3 Minimax搜索



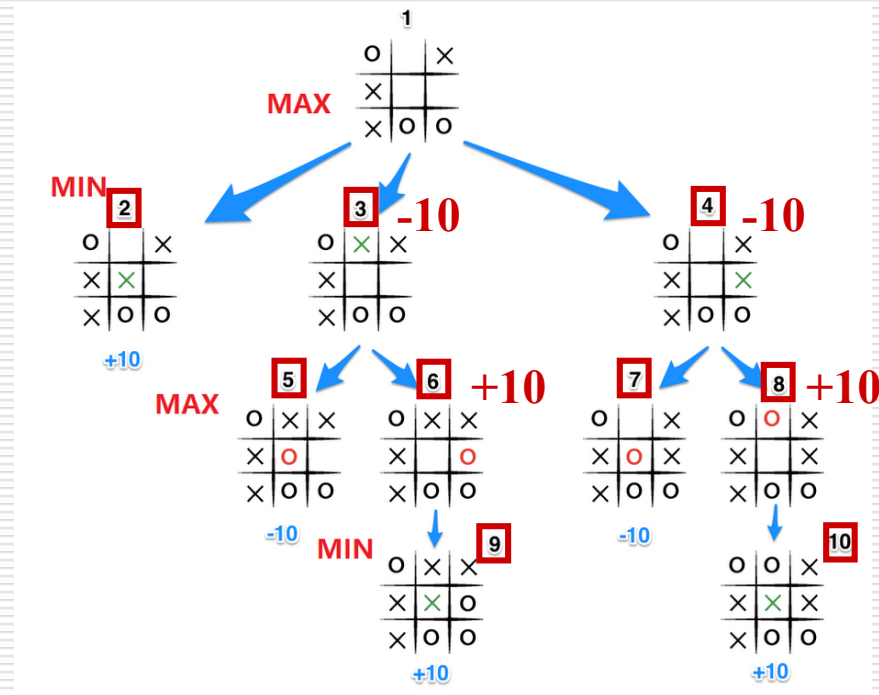
# 1.3 Minimax搜索



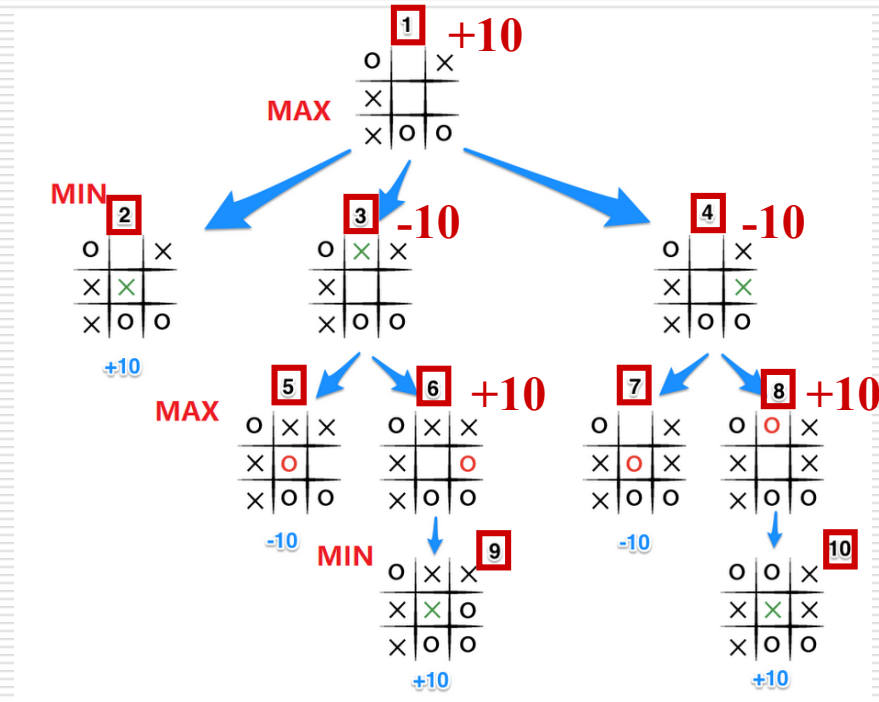
# 1.3 Minimax搜索



# 1.3 Minimax搜索



# 1.3 Minimax搜索

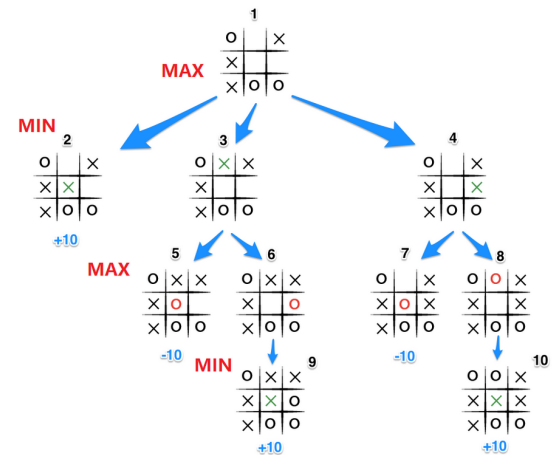


# 1.3 Minimax搜索

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

**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**  
     $v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(\text{RESULT}(s, a)))$   
**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**  
     $v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(\text{RESULT}(s, a)))$   
**return** *v*



## 1.4 Alpha-beta剪枝

- Minimax搜索：随着博弈的进行，必须检查的游戏状态的数目呈指数增长；
  - 我们只需要知道博弈过程所对应路径上的节点值；
- Alpha-beta剪枝：剪掉不可能影响决策的分支，尽可能地消除部分搜索树。
  - Max节点记录alpha值，Min节点记录beta值
  - Max节点的**alpha剪枝**：效益值  $\geq$  任何祖先Min节点的beta值
  - Min节点的**beta剪枝**：效益值  $\leq$  任何祖先Max节点的alpha值

# 1.4 Alpha-beta剪枝

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

---

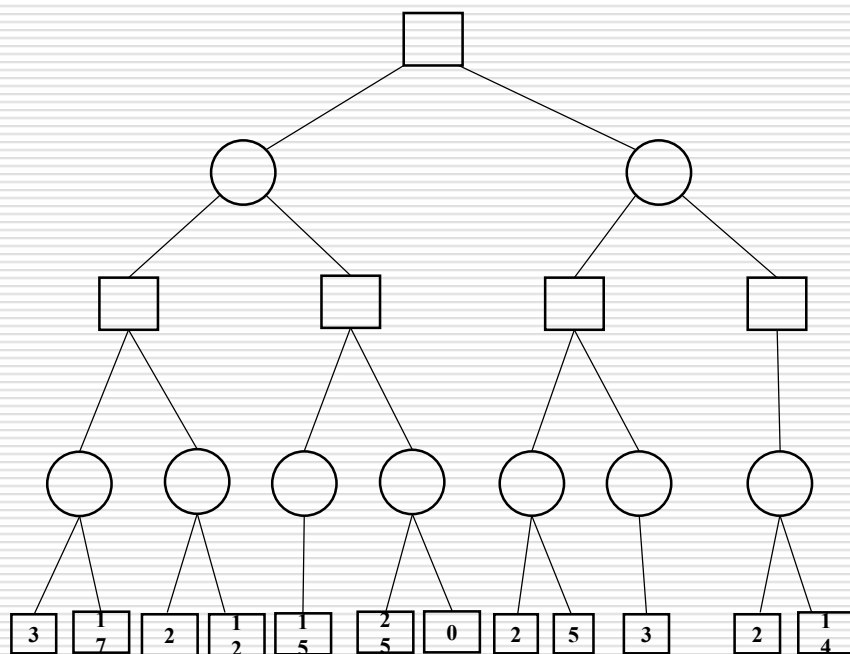
**function** MAX-VALUE( $state, \alpha, \beta$ ) **returns** a utility value  
    **if**  $\text{TERMINAL-TEST}(state)$  **then return**  $\text{UTILITY}(state)$   
     $v \leftarrow -\infty$   
    **for each**  $a$  **in**  $\text{ACTIONS}(state)$  **do**  
         $v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(\text{RESULT}(s,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

---

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



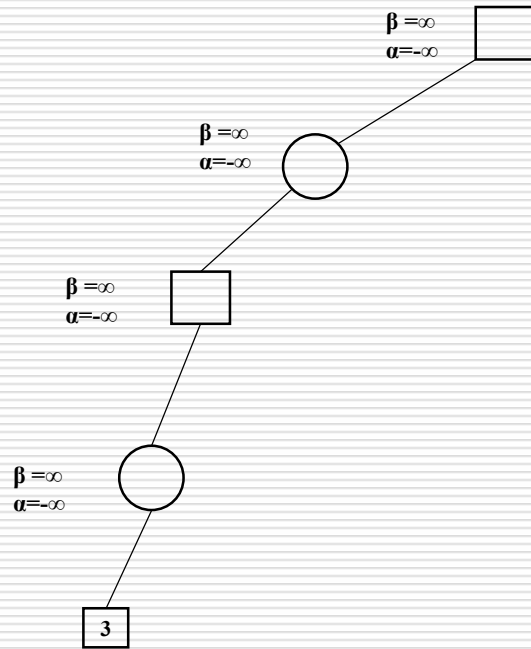
# 1.4 Alpha-beta剪枝



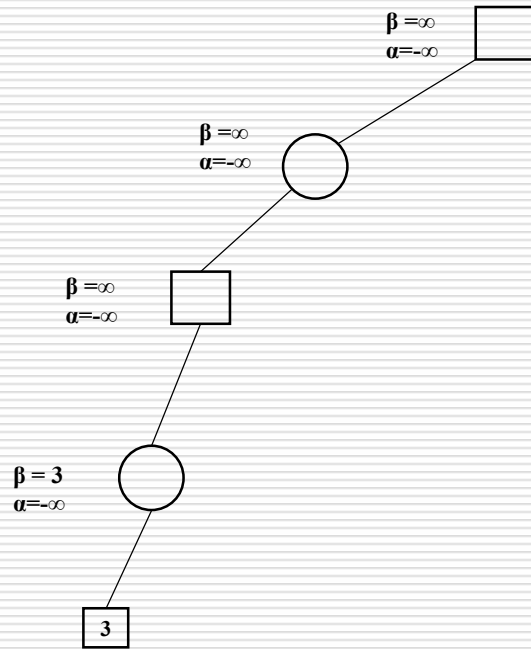
方形:Max节点  
圆形:Min节点

请写出对该博弈树进行搜索的过程，要求使用结合Alpha-beta剪枝的深度优先Minimax算法。

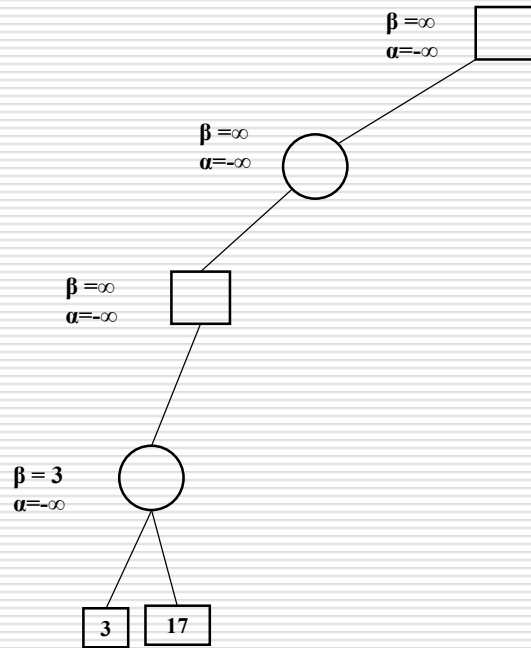
# 1.4 Alpha-beta剪枝



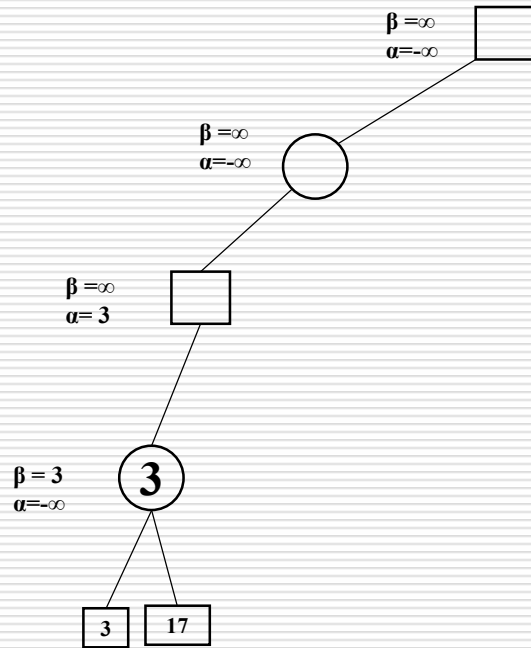
# 1.4 Alpha-beta剪枝



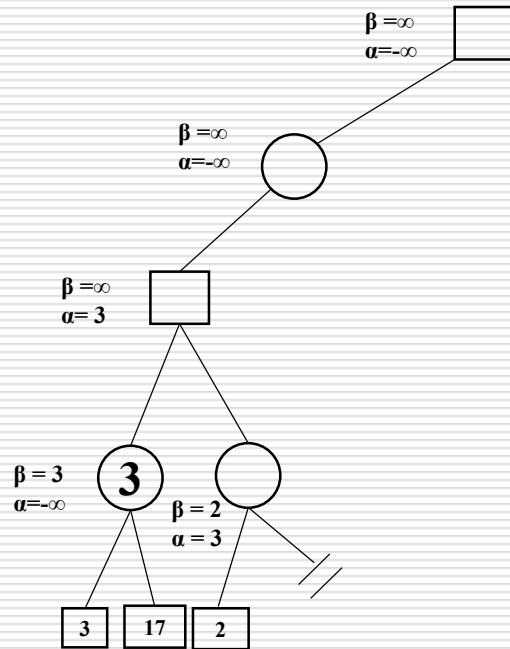
# 1.4 Alpha-beta剪枝



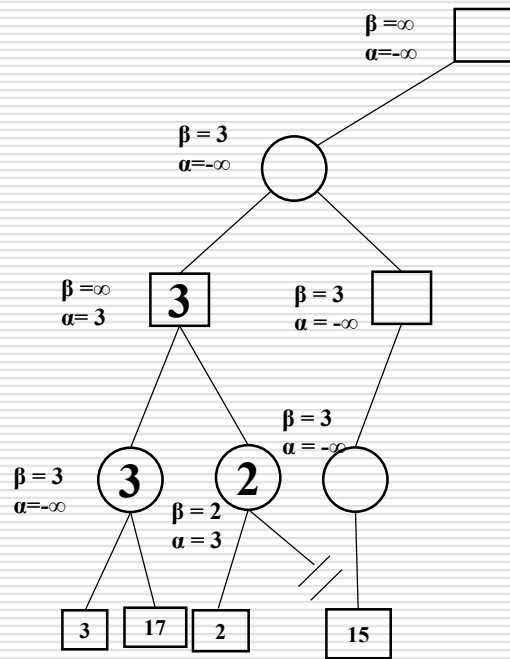
# 1.4 Alpha-beta剪枝



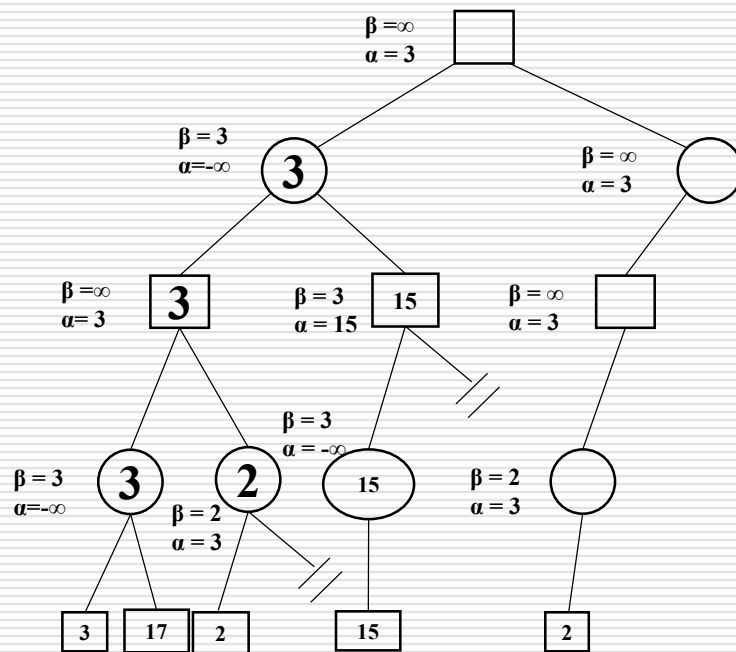
# 1.4 Alpha-beta剪枝



# 1.4 Alpha-beta剪枝

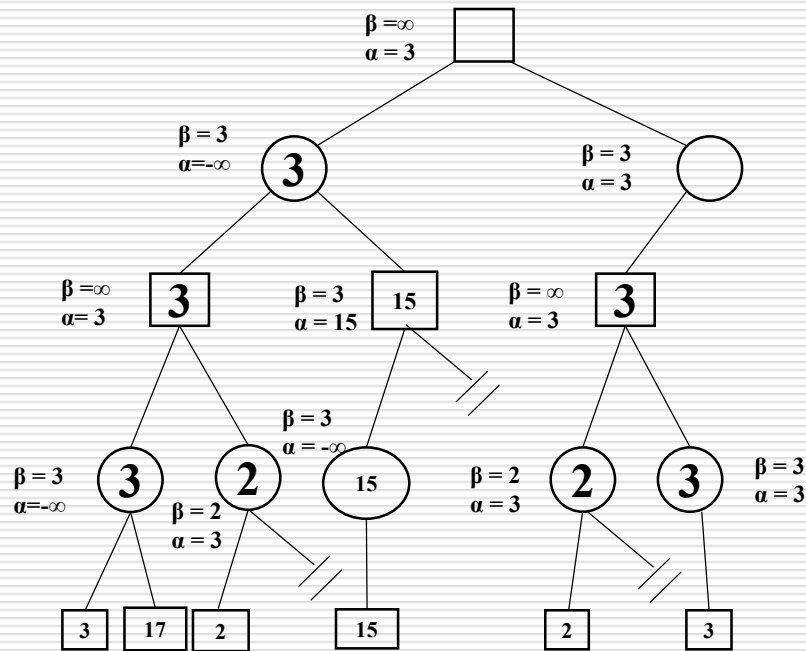


# 1.4 Alpha-beta剪枝





# 1.4 Alpha-beta剪枝



# 1.4 Alpha-beta剪枝

