对抗搜索：博弈
ADVERSARIAL SEARCH: GAME PLAYING

张玲玲
计算机学院
zhanglling@xjtu.edu.cn

主要内容

1. 博弈类型
   - 博弈的形式化
2. 博弈中的优化决策
   - 极小极大算法
   - α-β剪枝
   - 不完美的实时决策
3. 随机博弈
   - 期望极小极大值

博弈

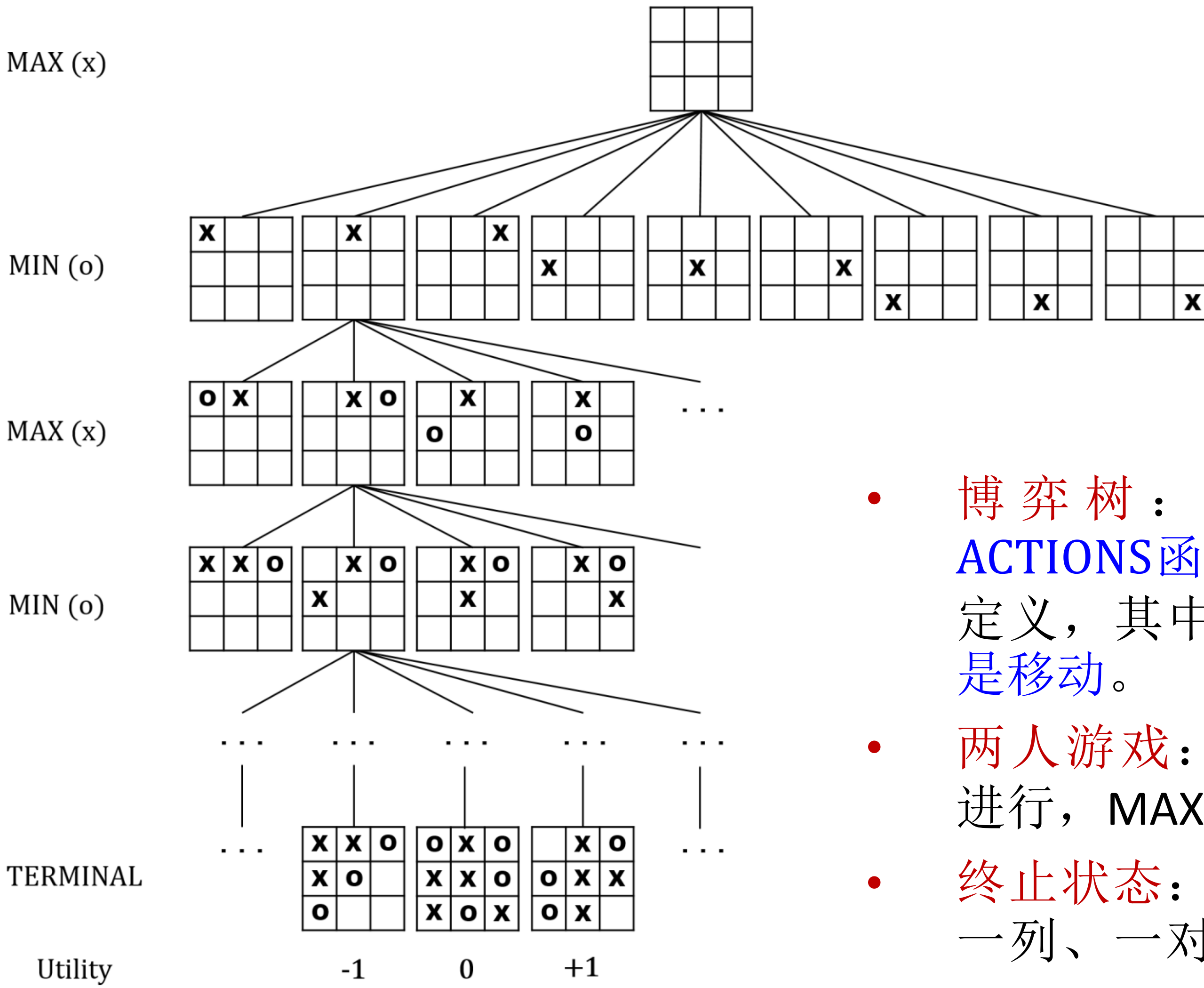
- 多Agent环境：每个Agent需要考虑其它Agent的行动及其对自身的影响。
- 竞争环境：每个Agent的目标之间是有冲突的。
- 对抗搜索（博弈）同时考虑多Agent和竞争环境。
- 数学中的博弈论，是经济学的一个分支，把多Agent环境看成是博弈，其中每个Agent都会受到其它Agent的显著影响，同时考虑竞争和合作环境。
- 人工智能中的“博弈”通常专指博弈论专家们称为有完整信息的、确定性的、轮流行动的、两个游戏者的零和游戏。
- 零和游戏指参与博弈的各方，在严格竞争下，一方的收益必然意味着另一方的损失，博弈各方的收益和损失相加总和永远为“零”（常量），双方不存在合作的可能。

博弈

- 两人参与的游戏：MAX和MIN。MAX先行，轮流出招，直到游戏结束。结束时优胜者加分，失败者罚分。
- S_0：初始状态，规范游戏开始时的情况。
- $\text{PLAYER}(s)$：处于状态s的游戏者，定义此时该谁行动。
- $\text{ACTIONS}(s)$：返回状态s下的合法移动集合。
- $\text{RESULT}(s,a)$：转移模型，返回状态s下行动a的结果状态。
- $\text{TERMINAL}-\text{TEST}(s)$：TRUE/FALSE，终止测试，游戏结束返回TRUE，否则返回FALSE。游戏结束的状态称为终止状态。
- $\text{UTILITY}(s,p)$：效用函数（目标函数或收益函数），定义游戏者p在终止状态s下的数值。
  - $\text{UTILITY}(s)$适用于两人游戏、零和博弈。因为：

$$\text{UTILITY}(s,p_1)=-\text{UTILITY}(s,p_2)$$

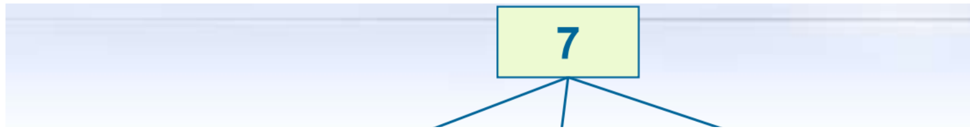
井字棋的部分博弈树



- 博弈树：由初始状态、ACTIONS函数和RESULT函数定义，其中结点是状态，边是移动。
- 两人游戏：MAX和MIN轮流进行，MAX下X，MIN下O。
- 终止状态：同一标志占一行、一列、一对角线或全部占满。

博弈树

MAX



MIN

MAX

MIN

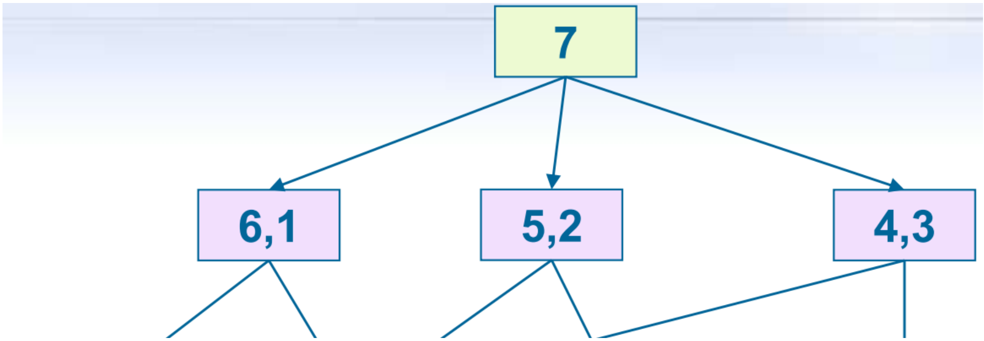
MAX

MIN

博弈树

MAX

MIN



MAX

MIN

MAX

MIN

博弈树

MAX

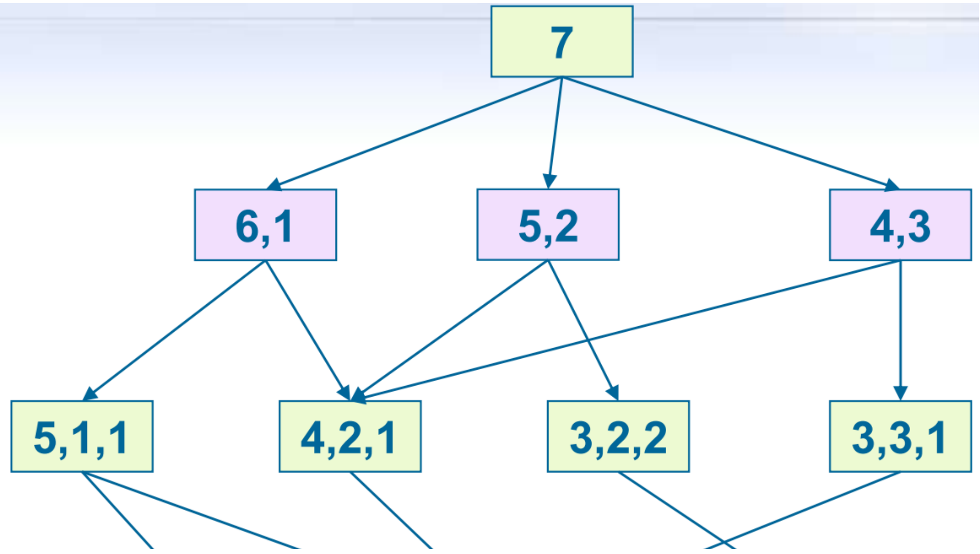
MIN

MAX

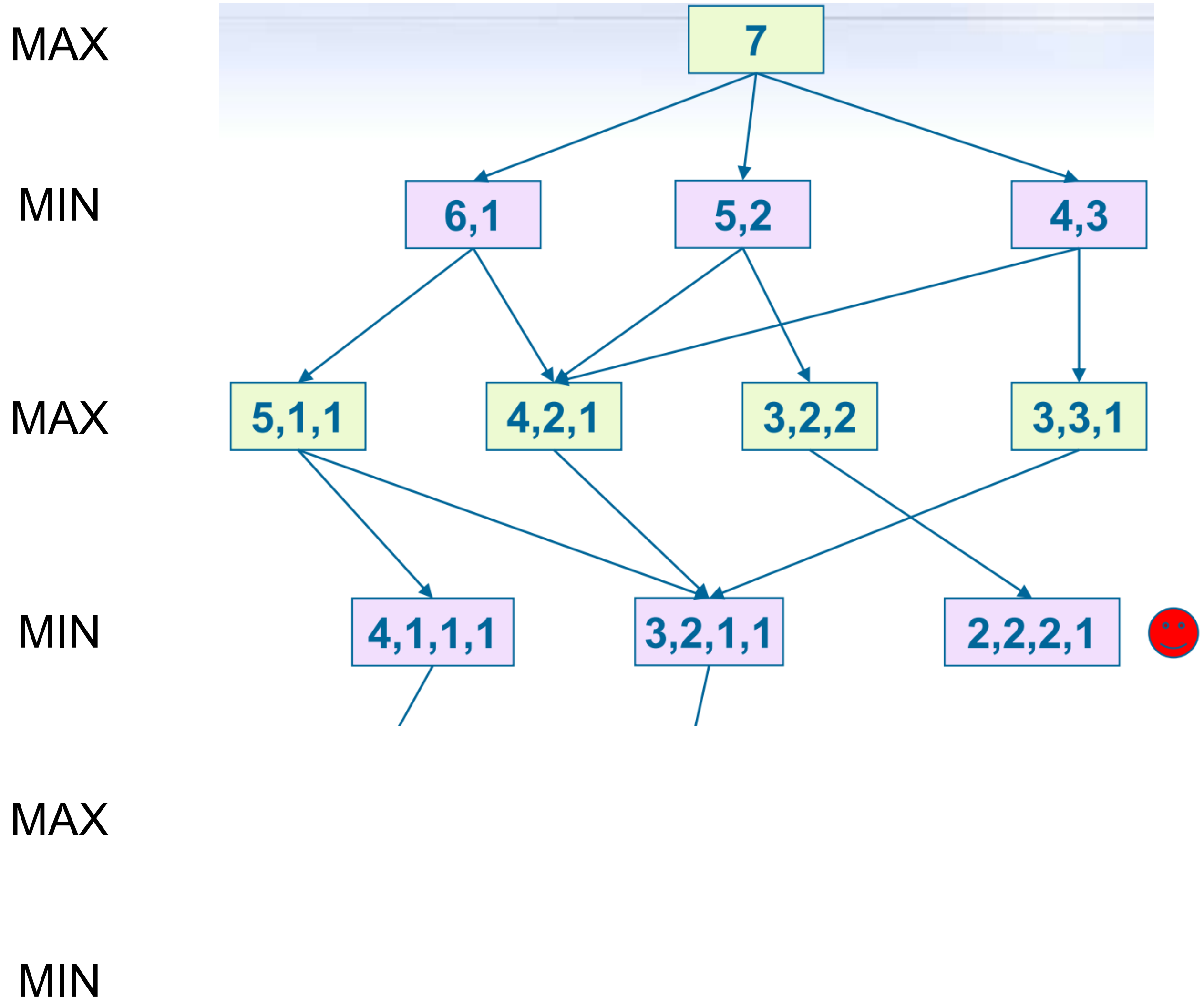
MIN

MAX

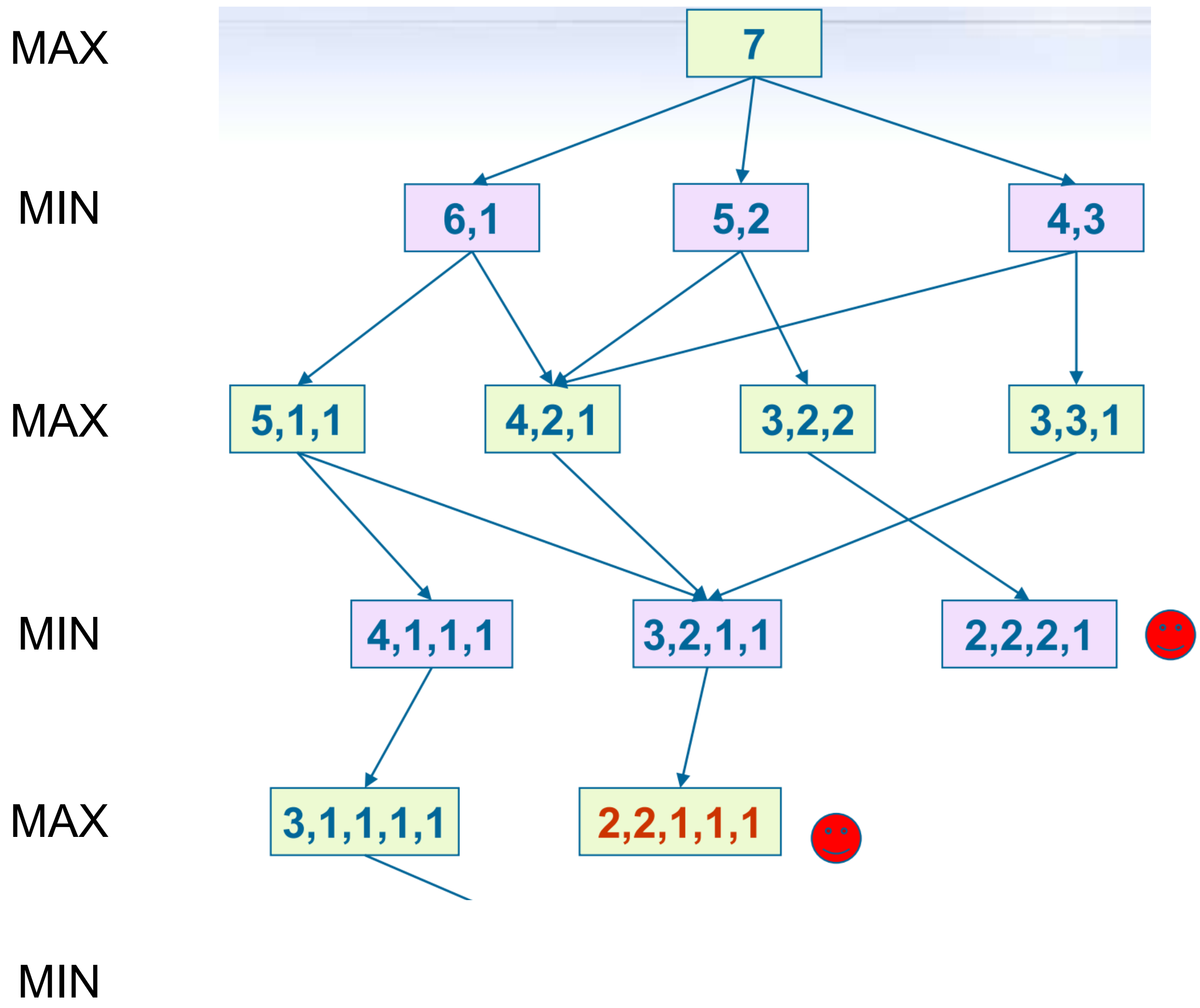
MIN



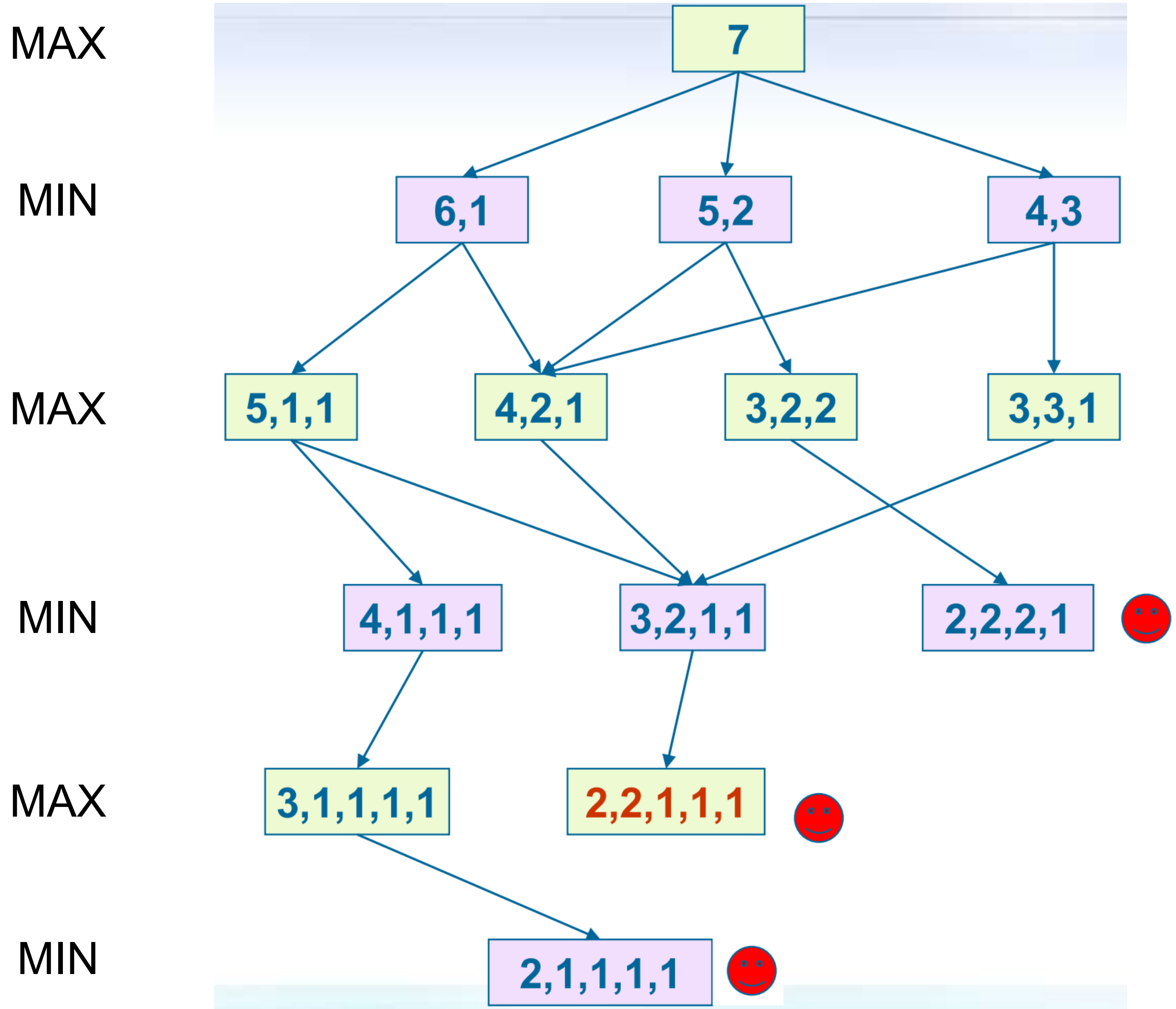
博弈树



博弈树


MAX
MIN
MAX
MIN
MAX
MIN
7
6,1
5,2
4,3
5,1,1
4,2,1
3,2,2
3,3,1
4,1,1,1
3,2,1,1
2,2,2,1
3,1,1,1,1
2,2,1,1,1

博弈树



MAX
MIN
MAX
MIN
MAX
MIN
7
6,1
5,2
4,3
5,1,1
4,2,1
3,2,2
3,3,1
4,1,1,1
3,2,1,1
2,2,2,1
3,1,1,1,1
2,2,1,1,1
2,1,1,1,1

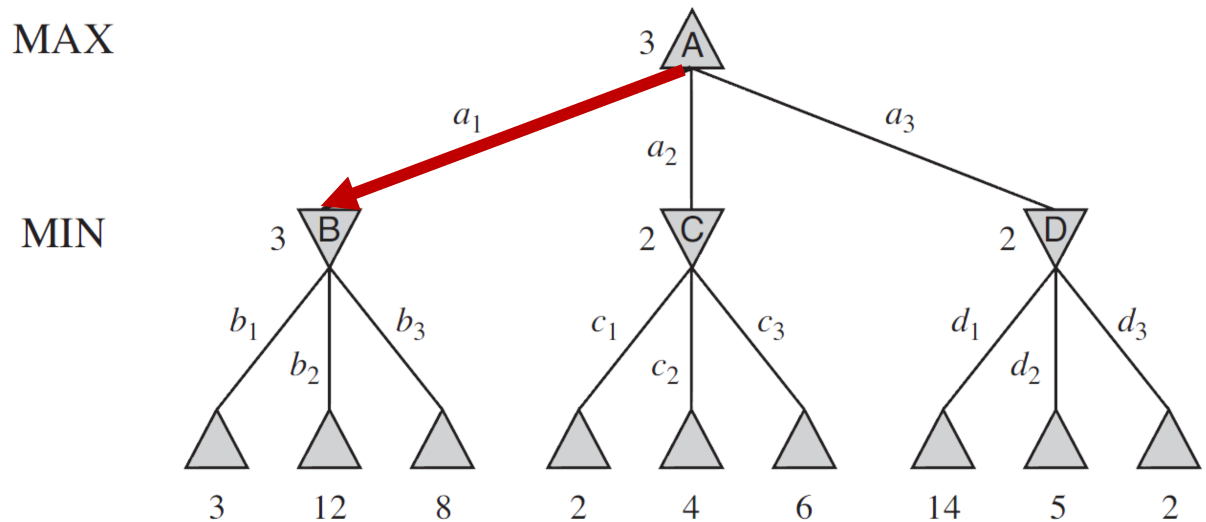
博弈中的优化决策

- 思想：给定一棵博弈树，最优策略可以通过检查每个结点的极小极大值（Minmax）来决定。
- 终止结点的极小极大值就是它自身的效用值。
- 对于给定的选择，MAX喜欢移动到有极大值的状态，而MIN喜欢移动到有极小值的状态。

- **结点*s*的极小极大值定义：**

$$
\textsc{Minimax}(s) =
\begin{cases}
\textsc{Utility}(s) & \text{if } \textsc{Terminal-Test}(s) \\
\max_{a \in Actions(s)} \textsc{Minimax}(\textsc{Result}(s,a)) & \text{if } \textsc{Player}(s) = \textsc{max} \\
\min_{a \in Actions(s)} \textsc{Minimax}(\textsc{Result}(s,a)) & \text{if } \textsc{Player}(s) = \textsc{min}
\end{cases}
$$

博弈中的优化决策



根结点A的极小极大决策：a_1是最优选择。

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

极小极大算法

```
function MINIMAX-DECISION(state) returns an action
    inputs: state, current state in game

    return the a in ACTIONS(state) maximizing MIN-VALUE(RESULT(a, state))
```

```
function MAX-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← −∞
    for a, s in SUCCESSORS(state) do v ← MAX(v, MIN-VALUE(s))
    return v
```

```
function MIN-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← ∞
    for a, s in SUCCESSORS(state) do v ← MIN(v, MAX-VALUE(s))
    return v
```

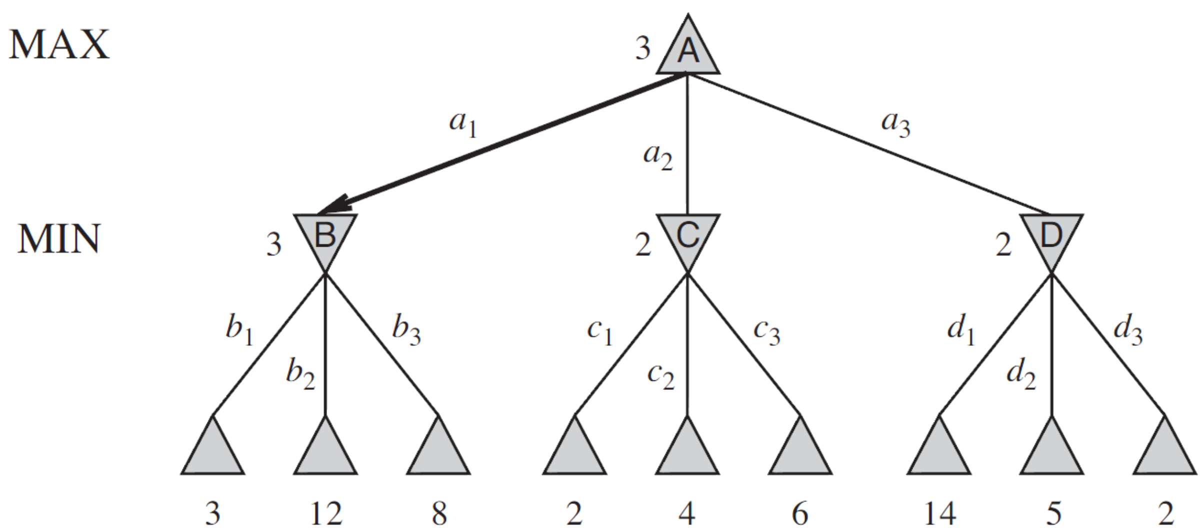
极小极大算法的性质

- 完备性：是，如果树是有限的。
- 最优性：是，当且仅当面对最优对手。
- 时间复杂度：$O(b^m)$。m是最大深度，b是最大分支因子。
- 空间复杂度： $O(bm)$（深度优先搜索）；或$O(m)$（如果算法一次生成一个动作）。

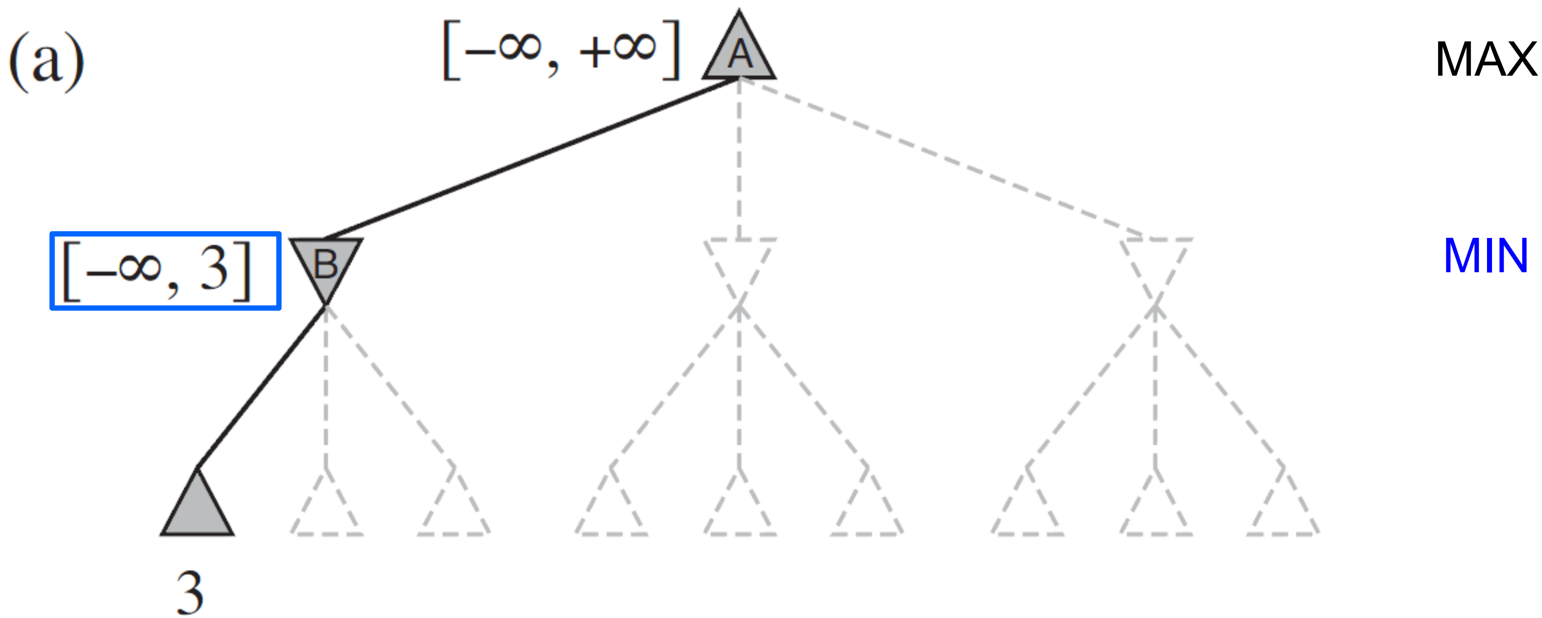
- 对于国际象棋，$b \approx 35$，$m \approx 100$ => 最优决策实际上很难解决。我们需要探索每条路径吗？

α-β剪枝

- 极小极大值搜索存在的问题：必须检查的游戏状态的数量随着博弈进行呈指数级增长。
- α-β剪枝的思想：尽可能消除部分搜索树——会剪掉不影响决策的分支，仍然返回和极小极大算法相同的结果。

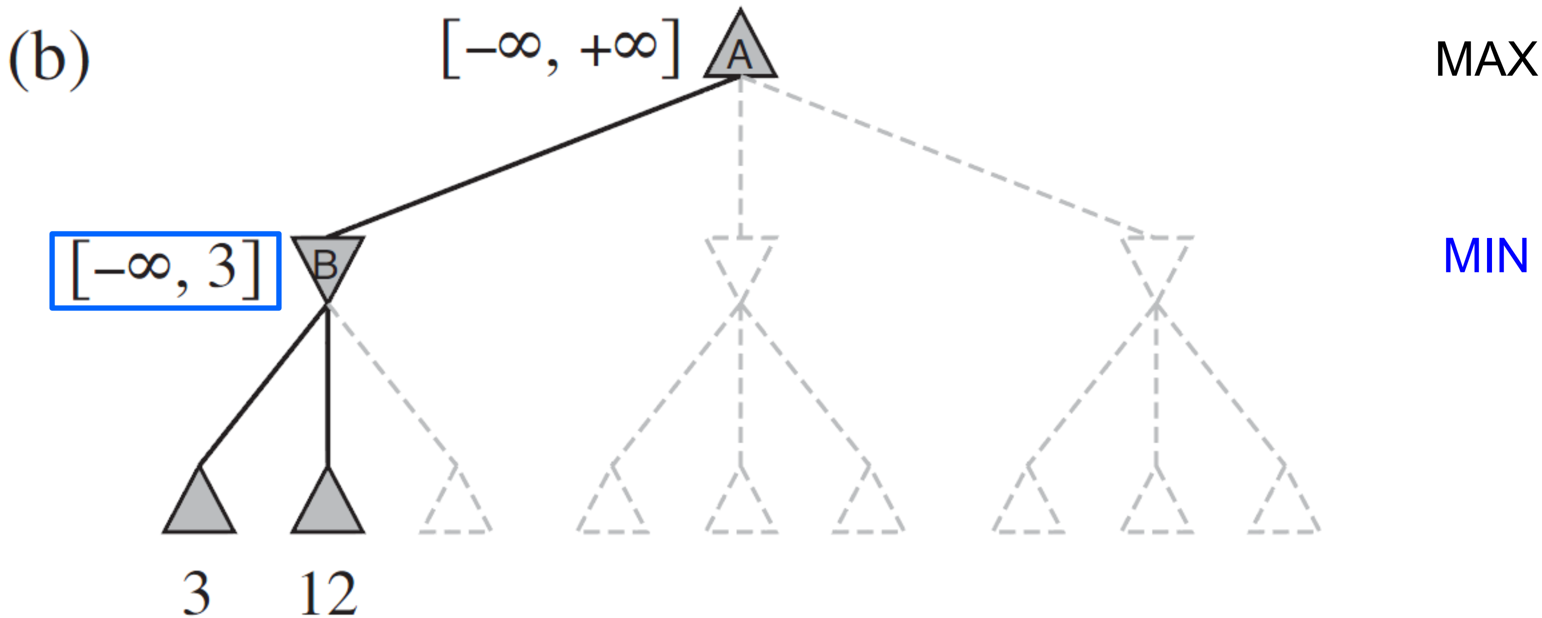


α-β剪枝举例



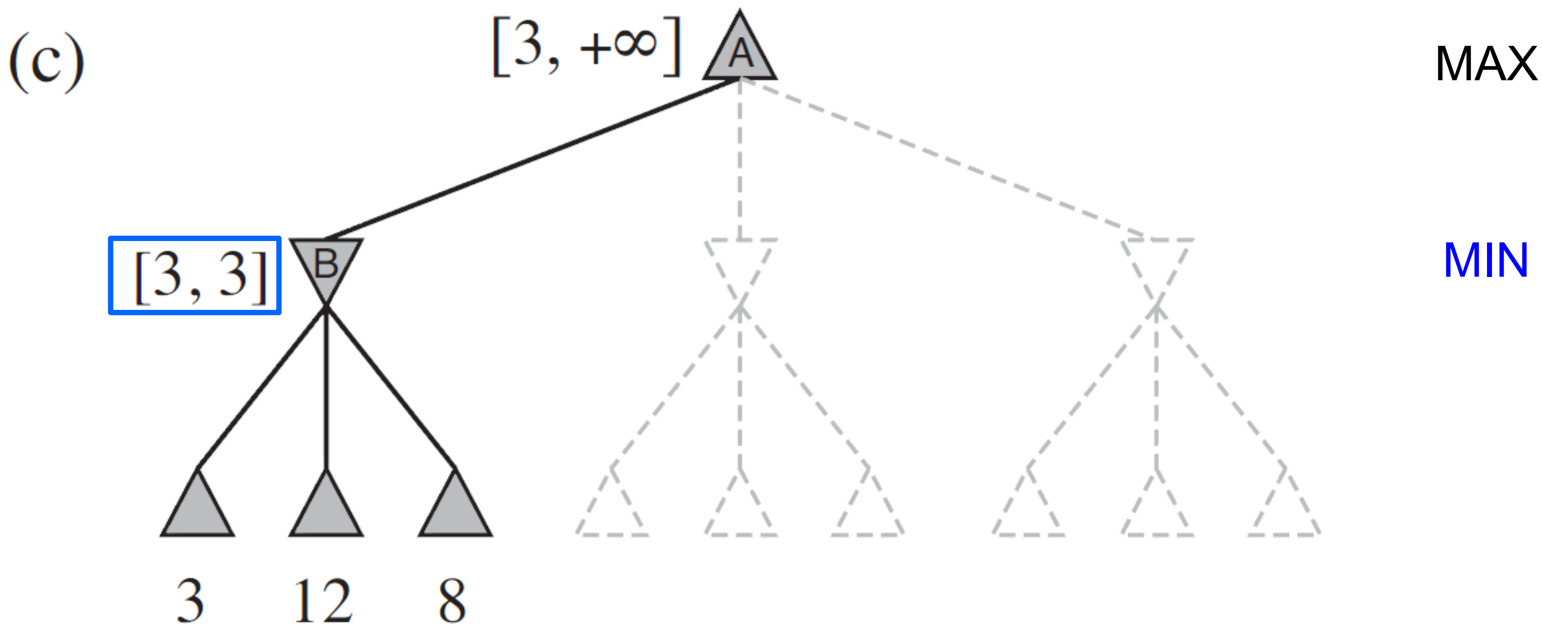
$$\text{MINIMAX}(s) = \begin{cases} \text{UTILITY}(s) & \text{if TERMINAL-TEST}(s) \\ \max_{a \in Actions(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if PLAYER}(s) = \text{MAX} \\ \boxed{\min_{a \in Actions(s)} \text{MINIMAX}(\text{RESULT}(s, a)) \quad \text{if PLAYER}(s) = \text{MIN}} \end{cases}$$

α-β剪枝举例



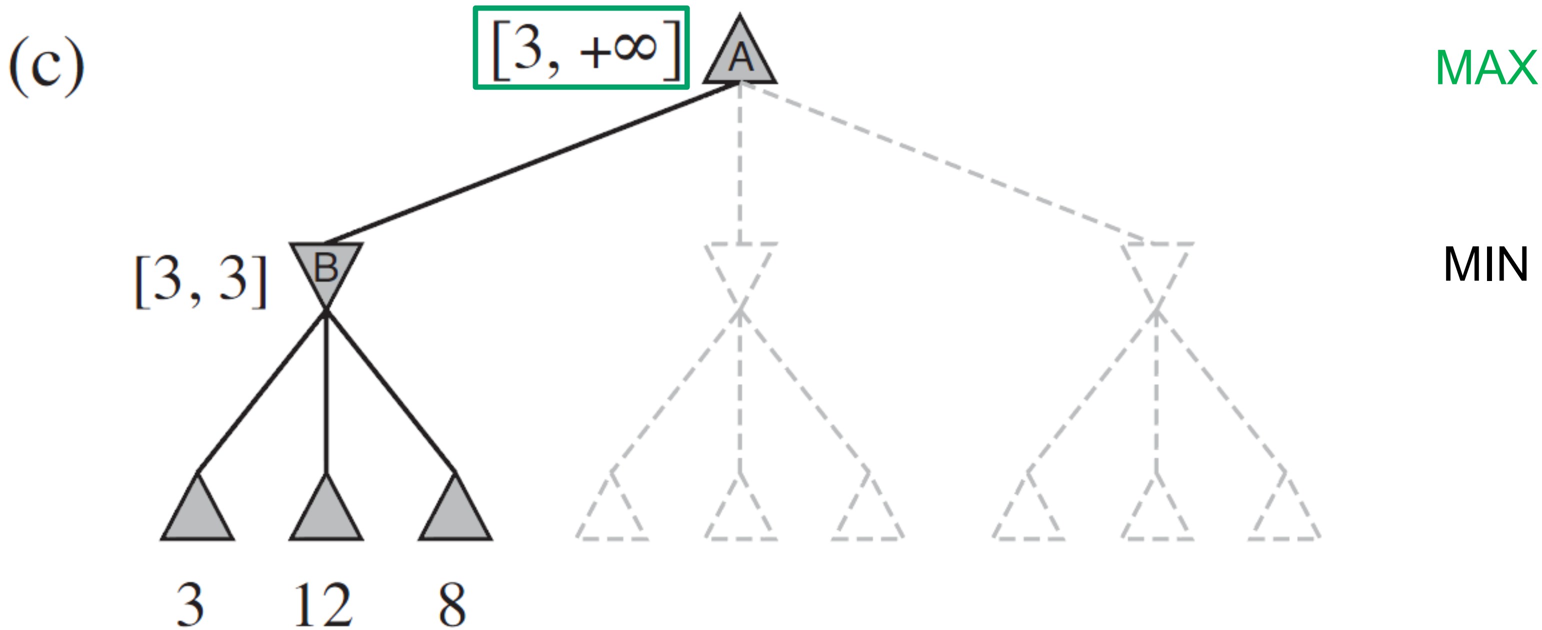
$$\text{MINIMAX}(s) = \begin{cases} \text{UTILITY}(s) & \text{if TERMINAL-TEST}(s) \\ \max_{a \in Actions(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MAX} \\ \boxed{\min_{a \in Actions(s)} \text{MINIMAX}(\text{RESULT}(s,a)) \quad \text{if PLAYER}(s) = \text{MIN}} \end{cases}$$

α-β剪枝举例



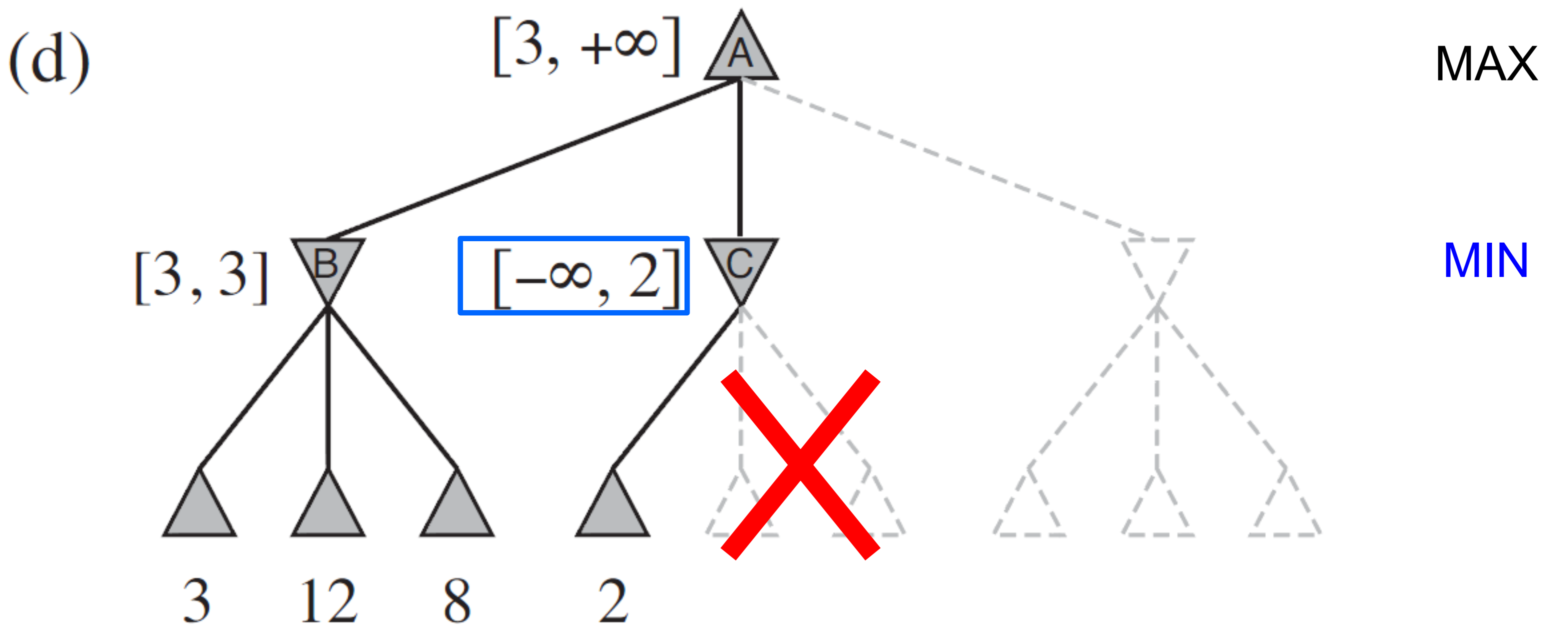
$$\text{MINIMAX}(s) = \begin{cases} \text{UTILITY}(s) & \text{if TERMINAL-TEST}(s) \\ \max_{a \in Actions(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MAX} \\ \boxed{\min_{a \in Actions(s)} \text{MINIMAX}(\text{RESULT}(s,a)) \quad \text{if PLAYER}(s) = \text{MIN}} \end{cases}$$

α-β剪枝举例



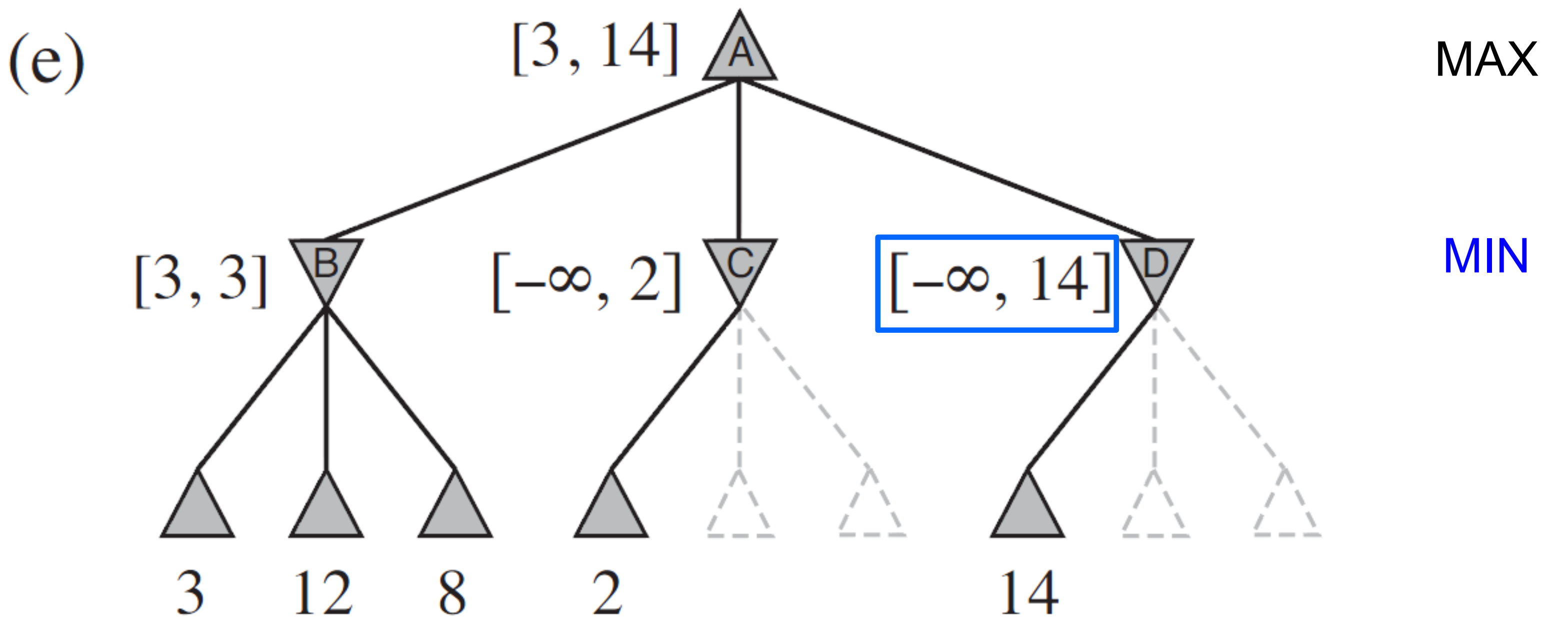
$$\textsc{Minimax}(s) = \begin{cases} \textsc{Utility}(s) & \text{if } \textsc{Terminal-Test}(s) \\ \max_{a \in Actions(s)} \textsc{Minimax}(\textsc{Result}(s, a)) & \text{if } \textsc{Player}(s) = \textsc{max} \\ \min_{a \in Actions(s)} \textsc{Minimax}(\textsc{Result}(s, a)) & \text{if } \textsc{Player}(s) = \textsc{min} \end{cases}$$

α-β剪枝举例



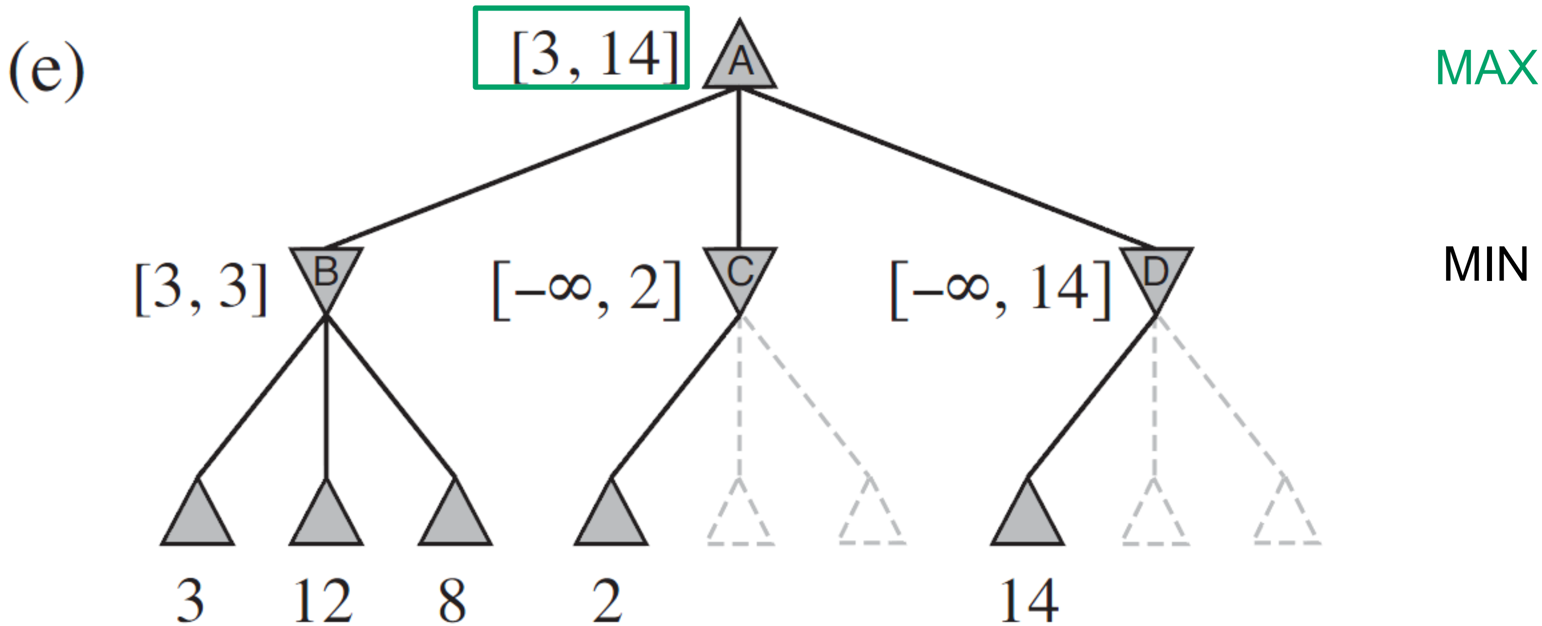
$$
\text{MINIMAX}(s) =
\begin{cases}
\text{UTILITY}(s) & \text{if TERMINAL-TEST}(s) \\
\max_{a \in Actions(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if PLAYER}(s) = \text{MAX} \\
\boxed{\min_{a \in Actions(s)} \text{MINIMAX}(\text{RESULT}(s, a)) \quad \text{if PLAYER}(s) = \text{MIN}}
\end{cases}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

α-β剪枝举例



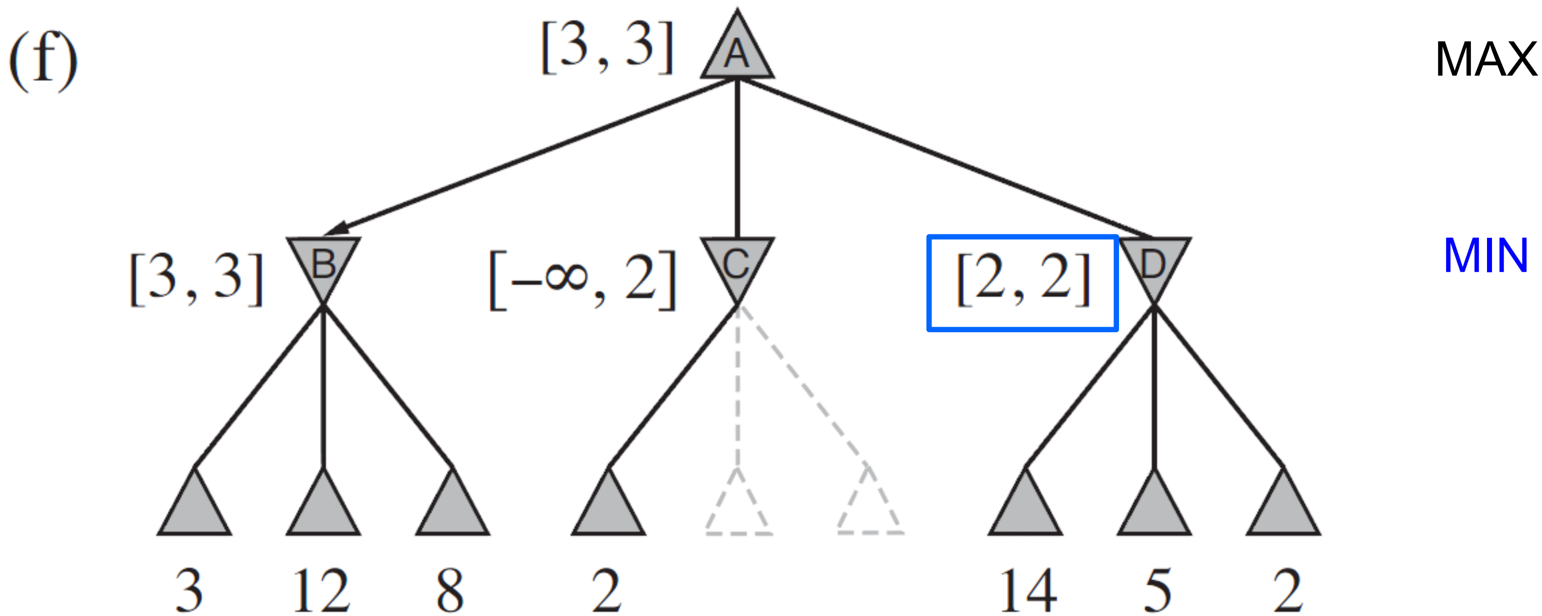
$$\text{MINIMAX}(s) = \begin{cases} \text{UTILITY}(s) & \text{if TERMINAL-TEST}(s) \\ \max_{a \in Actions(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MAX} \\ \boxed{\min_{a \in Actions(s)} \text{MINIMAX}(\text{RESULT}(s,a)) \quad \text{if PLAYER}(s) = \text{MIN}} \end{cases}$$

α-β剪枝举例



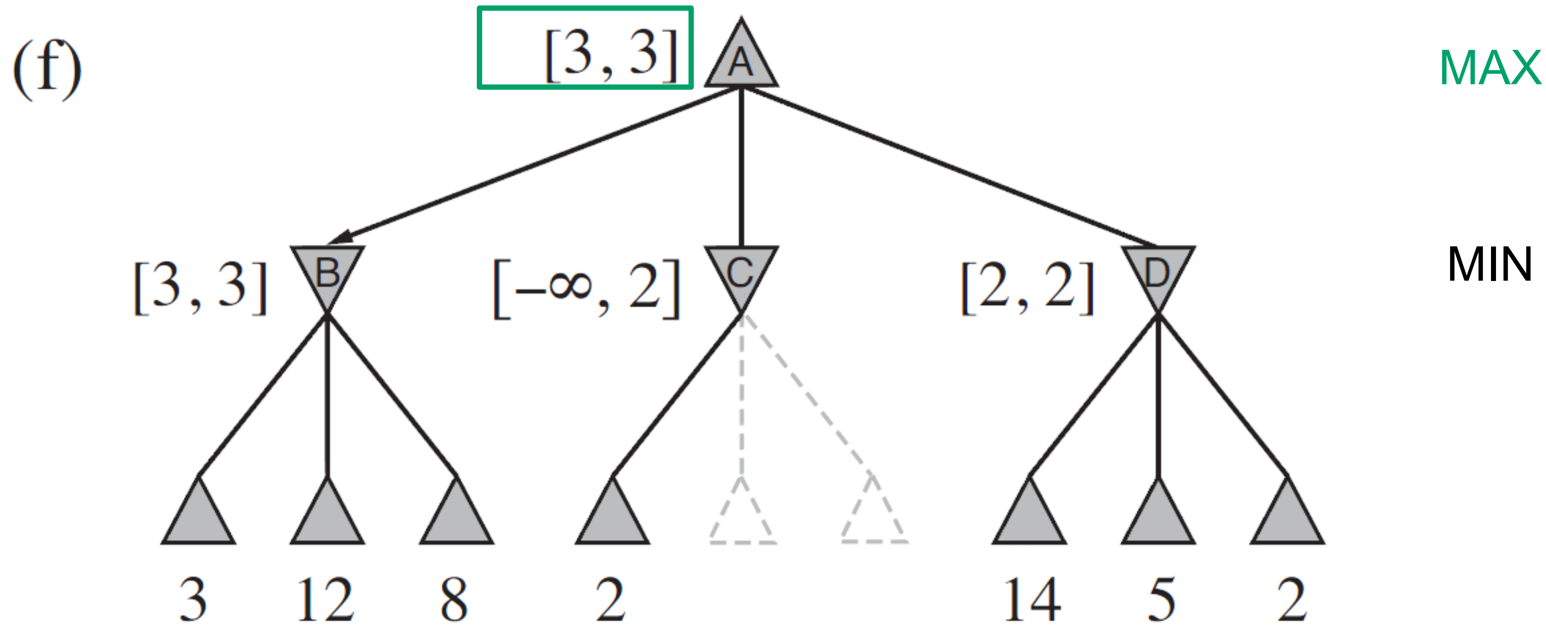
$$\text{MINIMAX}(s) = \begin{cases} \text{UTILITY}(s) & \text{if TERMINAL-TEST}(s) \\ \boxed{\max_{a \in Actions(s)} \text{MINIMAX}(\text{RESULT}(s, a)) \quad \text{if PLAYER}(s) = \text{MAX}} & \\ \min_{a \in Actions(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if PLAYER}(s) = \text{MIN} \end{cases}$$

α-β剪枝举例



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

α-β剪枝举例



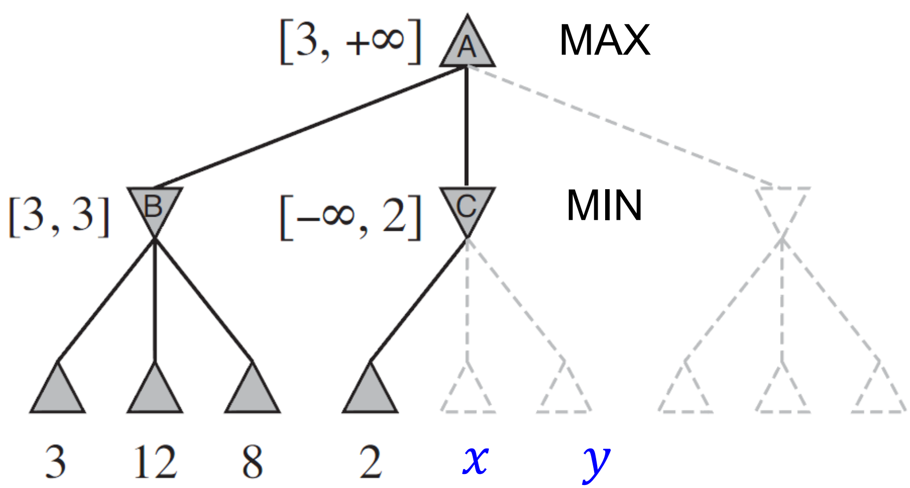
$$\text{MINIMAX}(s) = \begin{cases} \text{UTILITY}(s) & \text{if TERMINAL-TEST}(s) \\ \boxed{\max_{a \in Actions(s)} \text{MINIMAX}(\text{RESULT}(s, a)) \quad \text{if PLAYER}(s) = \text{MAX}} & \\ \min_{a \in Actions(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if PLAYER}(s) = \text{MIN} \end{cases}$$

α-β剪枝

- α-β剪枝的过程可以看做是对MINMAX公式的简化。
- 假设C结点的两个没有计算的叶节点的值是x和y。根节点A的值计算如下：

$$
\begin{aligned}
\mathrm{MINMAX}(root) &= \max(\min(3,12,8), \min(2,x,y), \min(14,5,2)) \\
&= \max(3, \min(2,x,y), 2) \\
&= \max(3, z, 2) \qquad \text{其中} z = \min(2,x,y) \leq 2 \\
&= 3
\end{aligned}
$$

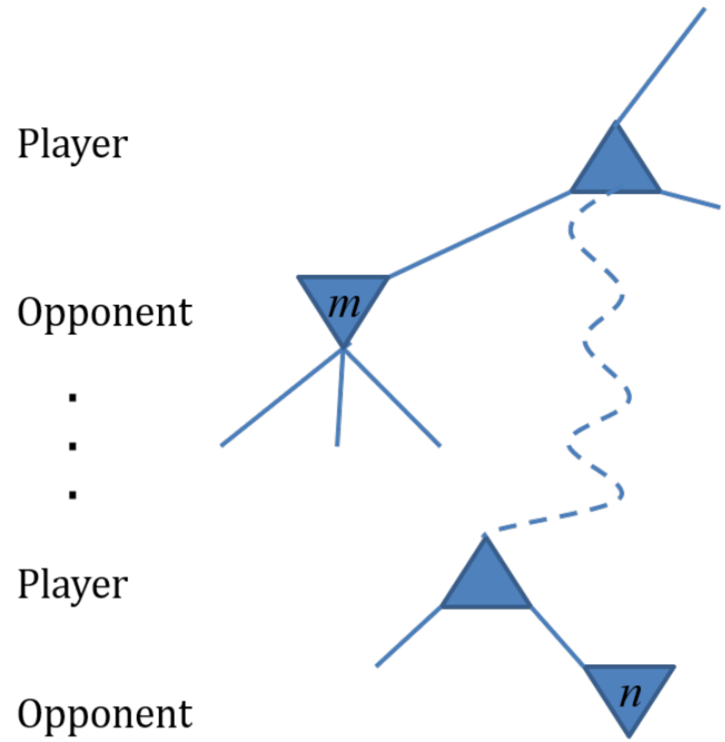
(d)



根结点的值以及因此做出的极小极大决策与被剪枝的叶节点x和y无关。

α-β剪枝的主要思想

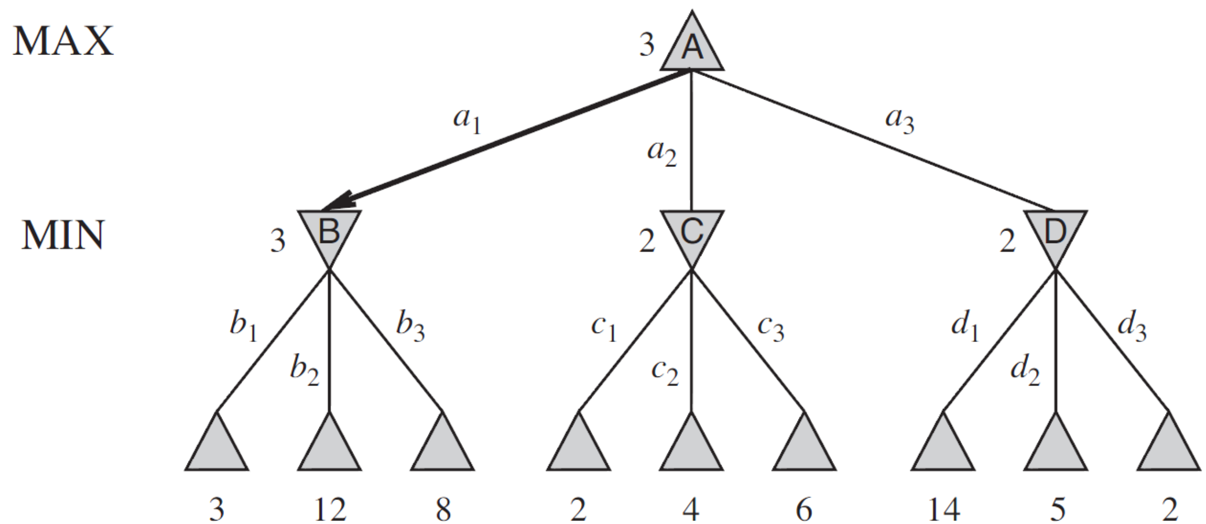
- α−β剪枝的一般原则：
  - 考虑在树中某处的结点n，选手选择移动到该结点。
  - 如果选手在n的父结点或者更上层的任何选择点有更好的选择m，那么在实际的博弈中永远不会到达n。
  - 一旦发现关于n的足够信息（检查它的某些后代），能够得出上述结论，就可以对其进行修剪。



如果对选手而言m比n好，那么就可以剪掉n。

α-β剪枝的主要思想

- 将玩家MAX的最大效用保持为α（最大下限，即$\geq \alpha$），并初始化为$-\infty$。
- 将玩家MIN的最小效用保持为β（最小上限$\leq \beta$），并初始化为$+\infty$。
- 仅扩展在窗口$[\alpha, \beta]$内的移动；否则将修剪其分支。
- 修剪不会影响解的质量。



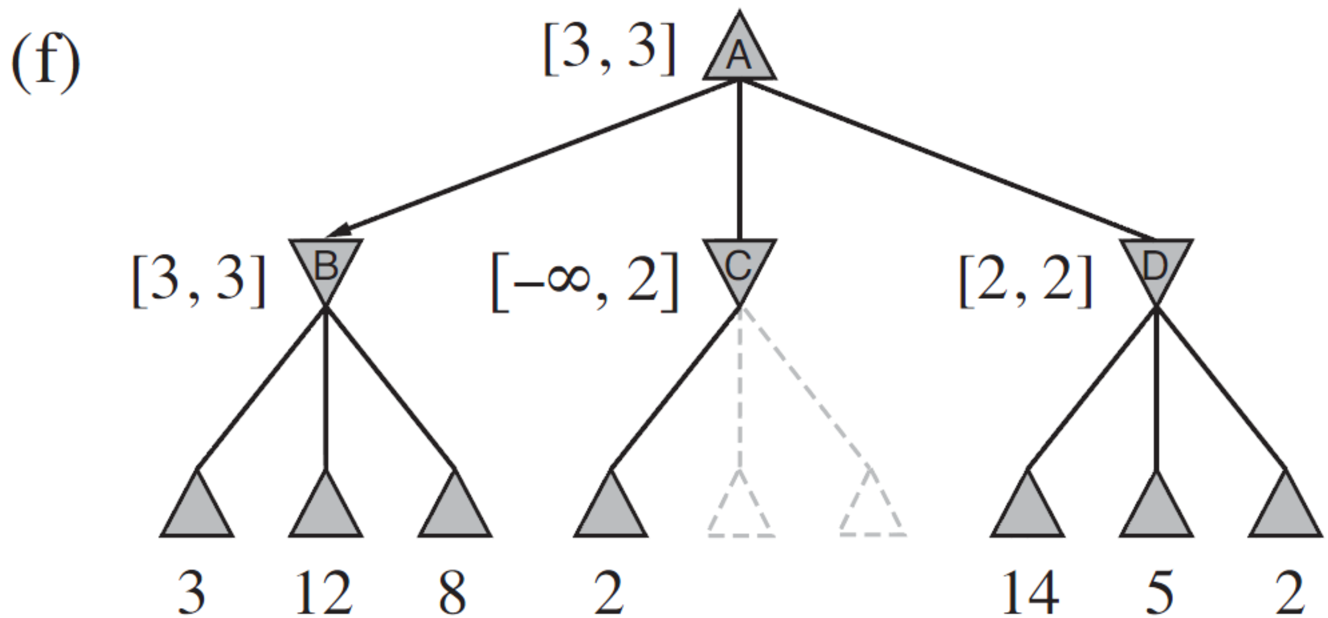
α-β剪枝的实现

ALPHABETA(s, α, β)

```
if TERMINAL-TEST(s)
    return UTILITY(s)
if PLAYER(s) == MAX
    v = −∞
    for each a ∈ ACTION(s)
        v = max(v, ALPHABETA(RESULT(s, a), α, β))
        if v ≥ β return v            超过最小上限
        α = max(α, v)
    return v
else
    v = ∞
    for each a ∈ ACTION(s)
        v = min(v, ALPHABETA(RESULT(s, a), α, β))
        if α ≥ v return v            低于最大下限
        β = min(β, v)
    return v
```

α-β剪枝的效率

- 很大程度上依赖于检查后继状态的顺序。
  - 好的移动顺序可以提高修剪的效率。



α-β剪枝的效率

- 很大程度上依赖于检查后继状态的顺序。
  - 好的移动顺序可以提高修剪的效率。
- 最坏的情况：不修剪 ---> $O(b^m)$。
- 最好的情况：始终先检查最佳移动。
  - 仍然需要检查第一个游戏者的每个移动。
  - 只需要检查第二个游戏者的一个移动（最好的后继）。
  - $O(b\times1\times b\times1\ \ldots) = O(b^{\frac{m}{2}})$
- 平均的情况：$O(b^{\frac{3m}{4}})$。（随机顺序）
- 非常简单的排序通常可以达到$O(b^{\frac{m}{2}})$。
  - 将有效分支因子减小为$\sqrt{b}$。
  - 在相同时间内，搜索的深度是极小极大搜索的两倍。

不完美的实时决策

- 将启发式评估函数用于搜索中的状态，把非终止结点转变为终止结点。也就是按两种方式对极大极小算法或者α-β算法进行修改：
  - 用估计棋局效用值的启发式评估函数EVAL取代效用函数UTILITY。
  - 用决策何时运用EVAL的截断测试（CUTOFF-TEST）取代终止测试。
- **结点s的启发式极小极大值定义（d是最大深度）：**

$$\text{H-MINIMAX}(s,d) = \begin{cases} \text{EVAL}(s) & \text{if CUTOFF-TEST}(s,d) \\ \max_{a \in Actions(s)} \text{H-MINIMAX}(\text{RESULT}(s,a), d+1) & \text{if PLAYER}(s) = \text{MAX} \\ \min_{a \in Actions(s)} \text{H-MINIMAX}(\text{RESULT}(s,a), d+1) & \text{if PLAYER}(s) = \text{MIN}. \end{cases}$$

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

不完美的实时决策

- 将启发式评估函数用于搜索中的状态，把非终止结点转变为终止结点。也就是按两种方式对极大极小算法或者α-β算法进行修改：
  - 用估计棋局效用值的启发式评估函数EVAL取代效用函数UTILITY。
  - 用决策何时运用EVAL的截断测试（CUTOFF-TEST）取代终止测试。
- **结点s的启发式极小极大值定义（d是最大深度）：**

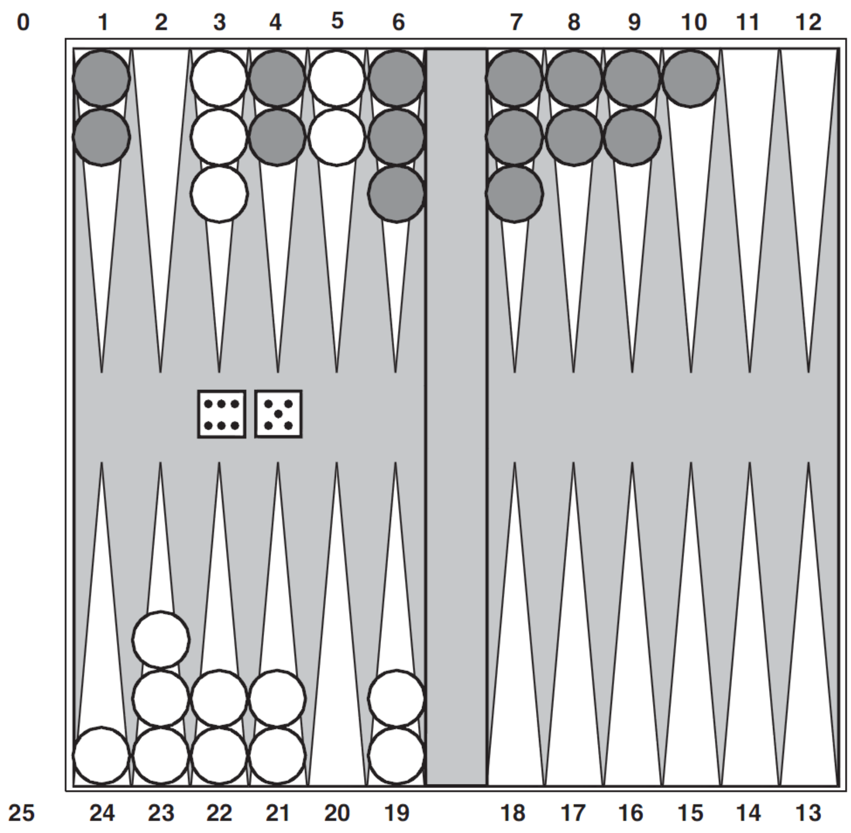
$$
\text{H-MINIMAX}(s,d) =
\begin{cases}
\text{EVAL}(s) & \text{if CUTOFF-TEST}(s,d) \\
\max_{a \in Actions(s)} \text{H-MINIMAX}(\text{RESULT}(s,a), d+1) & \text{if PLAYER}(s) = \text{MAX} \\
\min_{a \in Actions(s)} \text{H-MINIMAX}(\text{RESULT}(s,a), d+1) & \text{if PLAYER}(s) = \text{MIN.}
\end{cases}
$$

- 线性加权函数：$\text{EVAL}(s) = \sum_i w_i f_i(s)$，$w_i$是权值，$f_i(s)$是状态$s$的某个特征。

随机博弈

- 西洋双陆棋的目标：把自己的棋子全部移出棋盘者获胜。
- 运气和技巧：通过掷骰子决定合法移动。

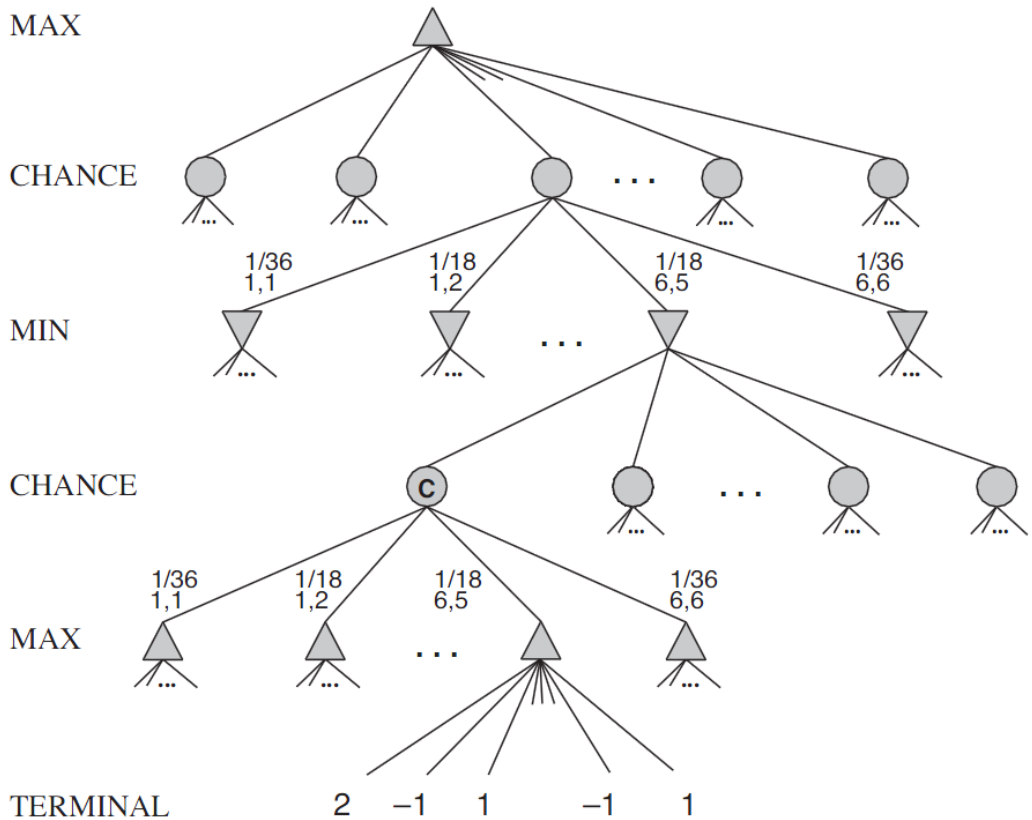
- 例子：白方掷出了6-5，有四种可能选择：
  - (5-10, 5-11)
  - (5-11, 19-24)
  - (5-10, 10-16)
  - (5-11, 11-16)



- 尽管白方知道自己的合法行棋，但不知道黑方会掷出多少及其合法行棋，因此无法构造像井字棋的标准搜索树。

随机博弈

- 除了MAX和MIN结点之外，必须引入机会结点(CHANCE)。
- 机会结点：圆圈
- 每个分支标记骰子数及其出现概率。
- 如何做出正确决策？
- 没有明确的极小极大值，只能计算棋局的期望极小极大值：机会结点所有可能结果的平均值。



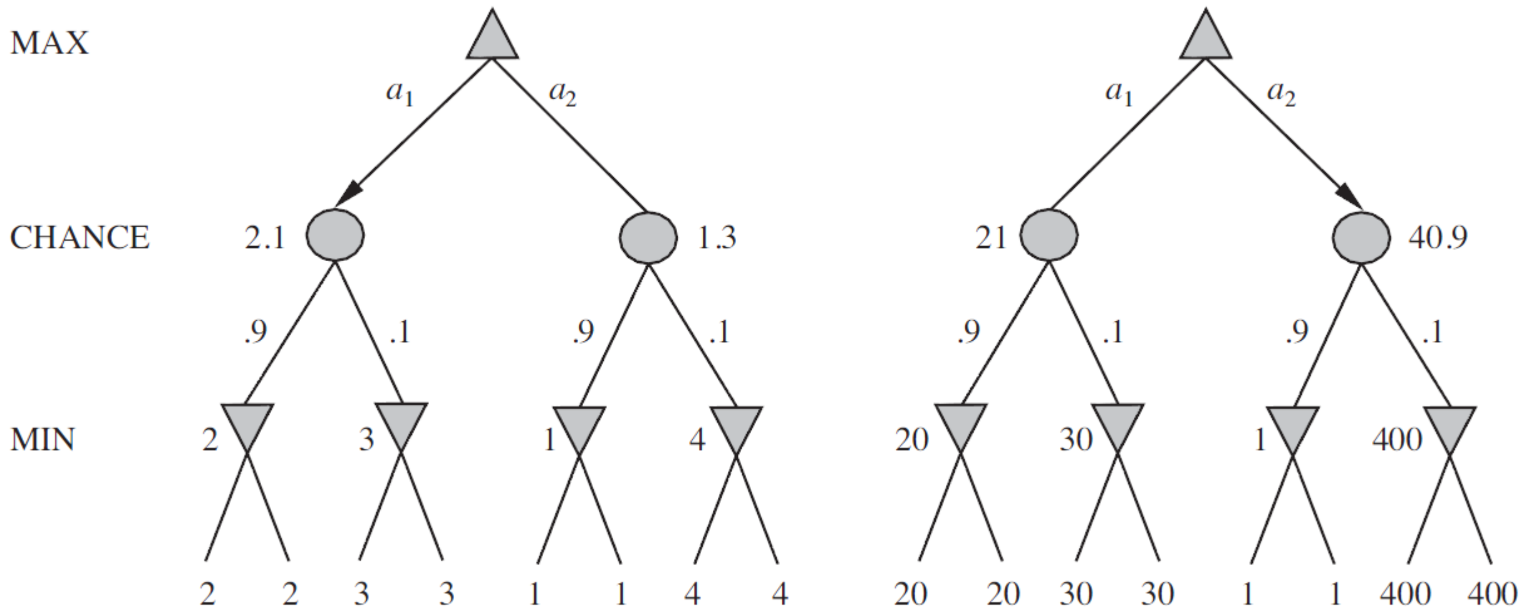
期望极小极大

- 把确定性博弈中的极小极大值一般化为包含机会结点的博弈的期望极小极大值，其中MAX和MIN结点的使用和以前完全一样。
- 结点s的期望极小极大值计算如下：

$$\textsc{Expectiminimax}(s) = \begin{cases} \textsc{Utility}(s) & \text{if } \textsc{Terminal-Test}(s) \\ \max_a \textsc{Expectiminimax}(\textsc{Result}(s,a)) & \text{if } \textsc{Player}(s) = \textsc{max} \\ \min_a \textsc{Expectiminimax}(\textsc{Result}(s,a)) & \text{if } \textsc{Player}(s) = \textsc{min} \\ \sum_r P(r)\textsc{Expectiminimax}(\textsc{Result}(s,r)) & \text{if } \textsc{Player}(s) = \textsc{chance} \end{cases}$$

$$\textsc{Minimax}(s) = \begin{cases} \textsc{Utility}(s) & \text{if } \textsc{Terminal-Test}(s) \\ \max_{a \in Actions(s)} \textsc{Minimax}(\textsc{Result}(s,a)) & \text{if } \textsc{Player}(s) = \textsc{max} \\ \min_{a \in Actions(s)} \textsc{Minimax}(\textsc{Result}(s,a)) & \text{if } \textsc{Player}(s) = \textsc{min} \end{cases}$$

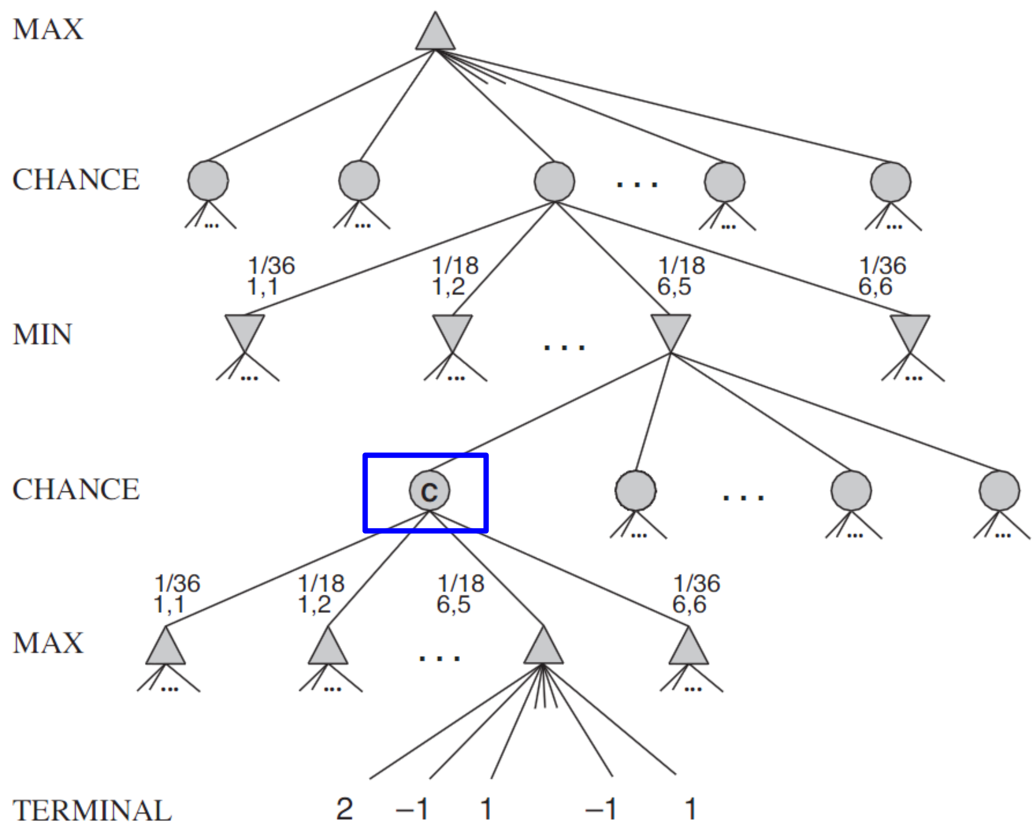
机会博弈中的评估函数



- 和极大极小值一样，期望极大极小值的近似估计可以通过在某结点截断搜索并对每个叶节点计算其评估函数来进行。
  - 评估函数：对好的行棋给予高分。
  - 叶节点的评估函数值范围是[1,2,3,4]，a_1是最佳行棋。
  - 叶节点的评估函数值范围是[1,20,30,400]，a_2是最佳行棋。

期望极小极大的性能

- $\alpha-\beta$剪枝不适用于期望极小极大值的MAX/MIN结点。
- $\alpha-\beta$剪枝仍然适用于机会结点。
- 时间复杂度：$O(b^m) \rightarrow O(b^m n^m)$，其中$n$是不同掷骰子结果的数量。



小结

1. 博弈类型
   - 博弈的形式化
2. 博弈中的优化决策
   - 极小极大算法：适用于两人零和博弈
   - α-β剪枝：相同时间内，使搜索更深入。
   - 不完美的实时决策：启发式评估函数EVAL取代效用函数UTILITY，截断搜索
3. 随机博弈
   - 期望极小极大值：引入机会结点