# 博弈树搜索

陆俞因

2023.04.04

# 博弈树搜索


➢理论课内容回顾

- （问题描述）两玩家零和博弈问题
- （数学形式化）博弈树
- （解决方案）Minimax搜索
- （优化方案）Alpha-beta剪枝

➢实验任务

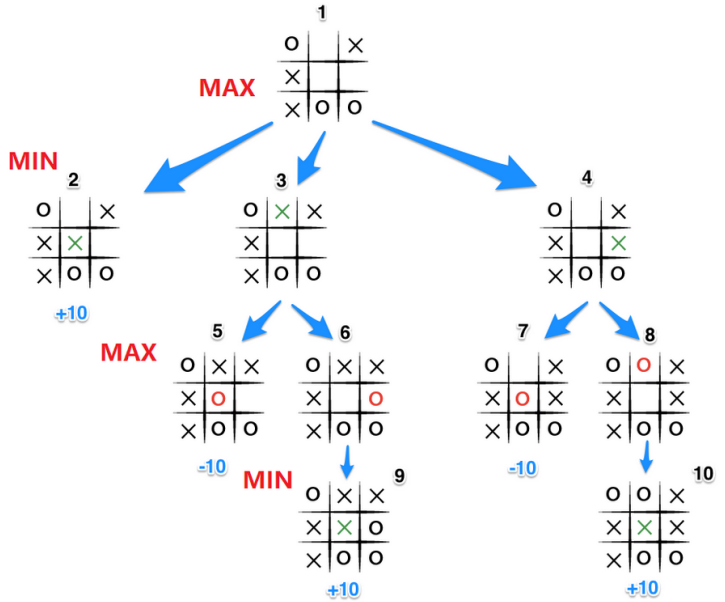
# 两玩家零和博弈问题

- 两名玩家轮流行动，进行博弈
- 有限的：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）

# 博弈树：基本概念回顾

- 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）。
- 双方逐层交替扩展节点：两个玩家的行动轮流出现。
  - 在博弈树中，游戏“状态”是一个（状态-玩家）对
- 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“效益值”。

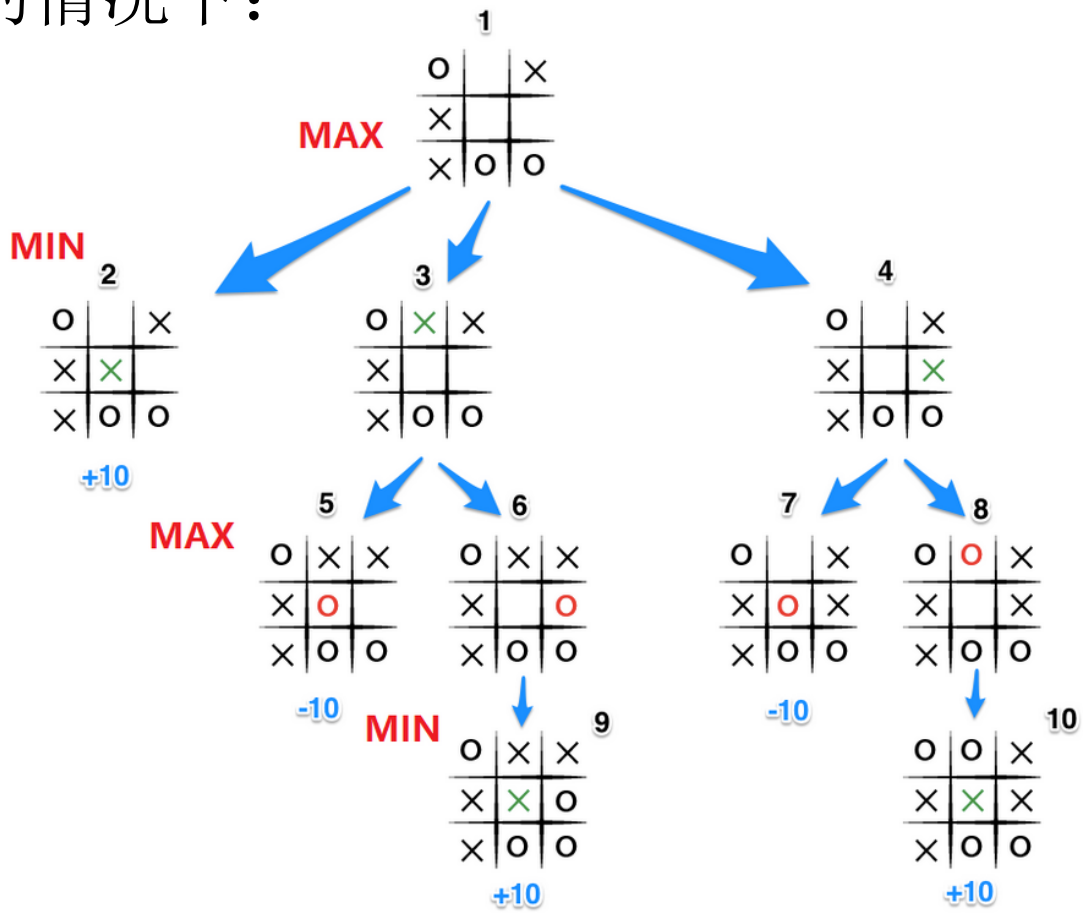
# Minimax搜索



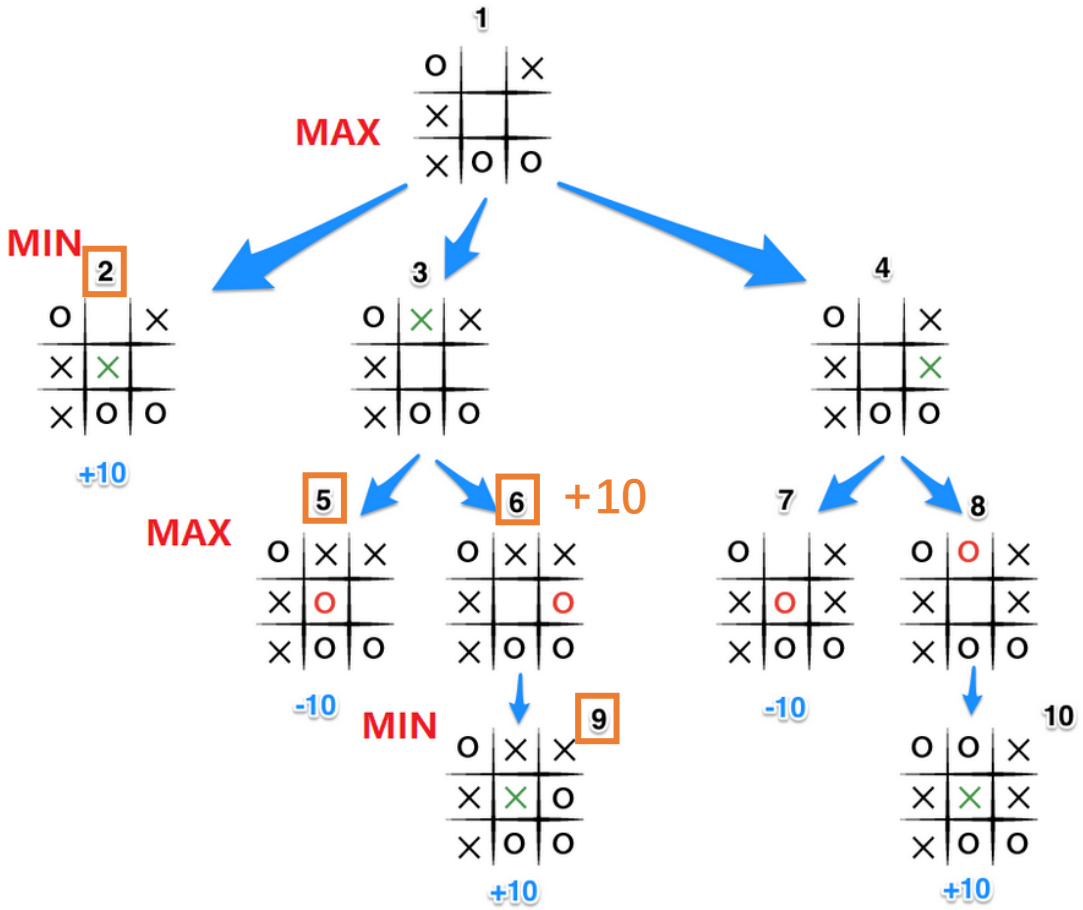

- 假设：
  - 玩家A和玩家B的行动逐层交替；
  - A和B的利益关系对立，即假设A要使分数更大，B就要使分数更小；
  - A和B均采取最优策略。
- Minimax搜索：找到博弈树中内部节点的“效益值”，其中Max节点（A）的每一步扩展要使收益最大，Min节点（B）的扩展要使收益最小。

# Minimax搜索

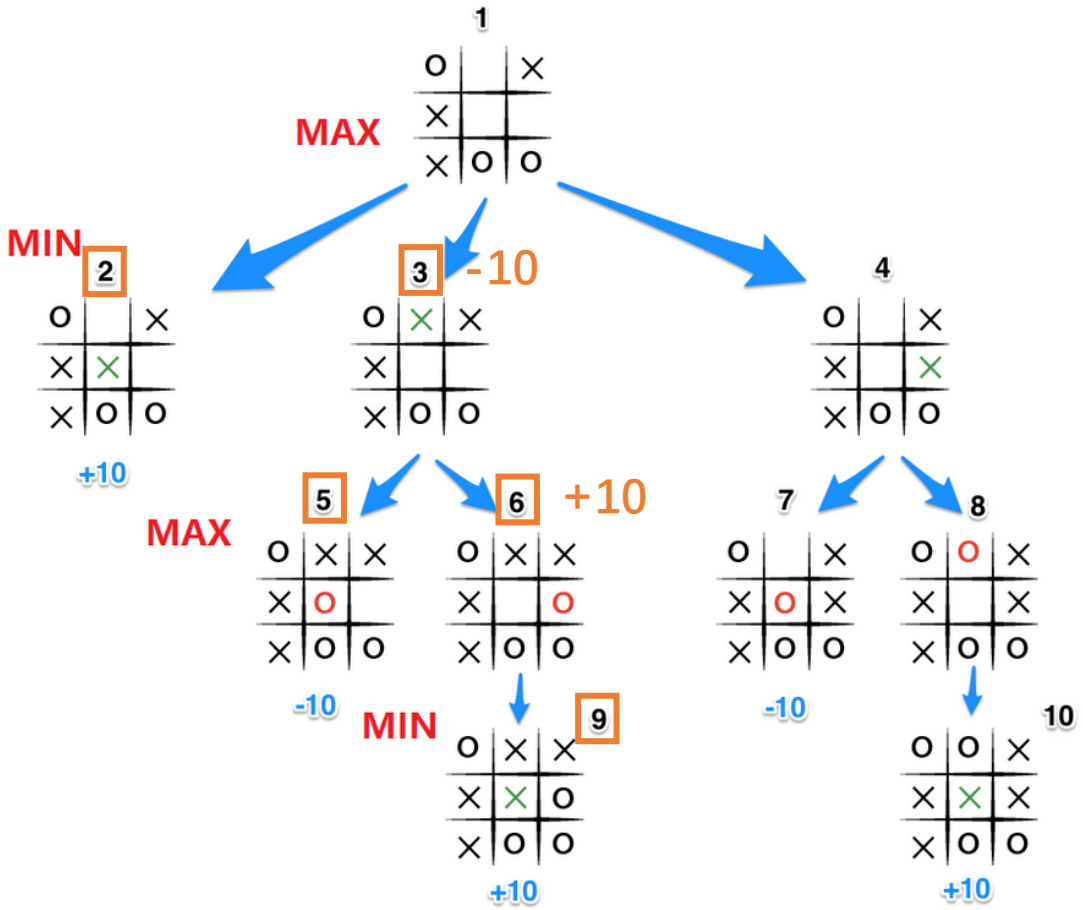
在完整博弈树已知的情况下：



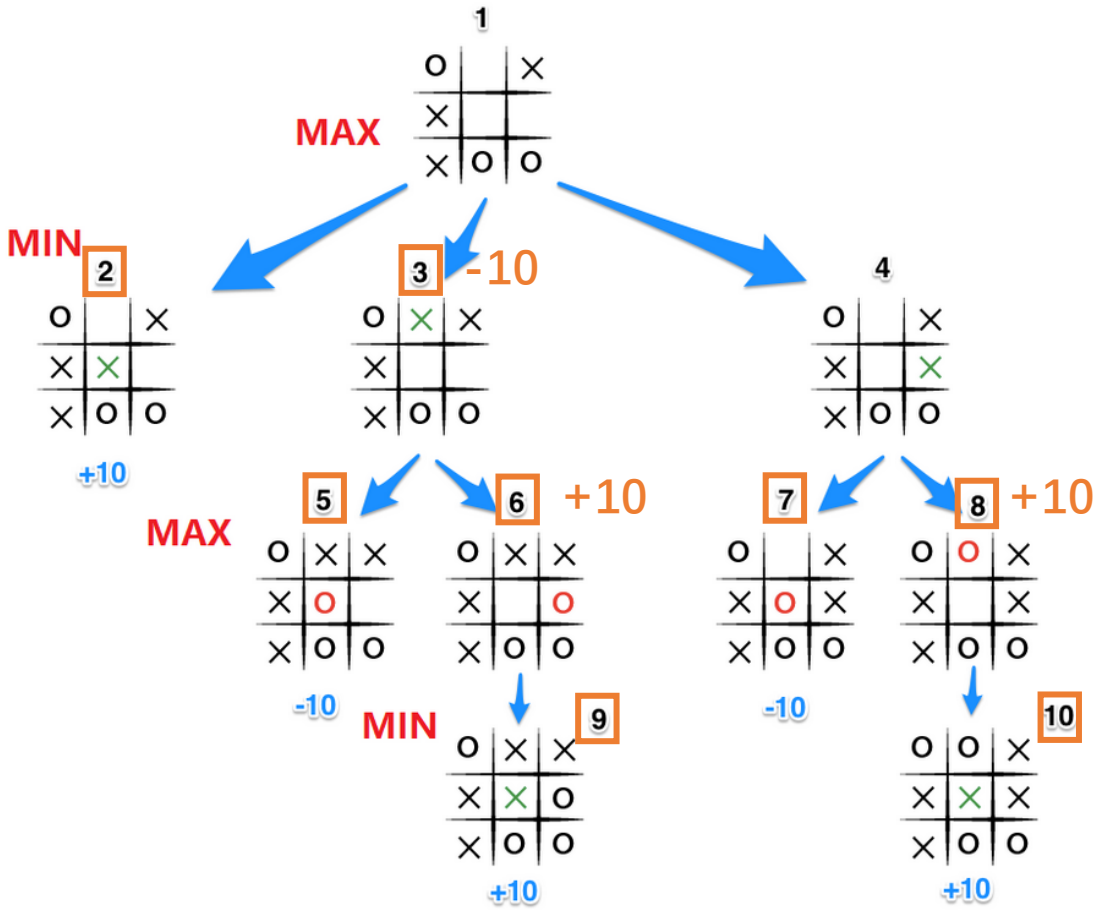
# Minimax搜索



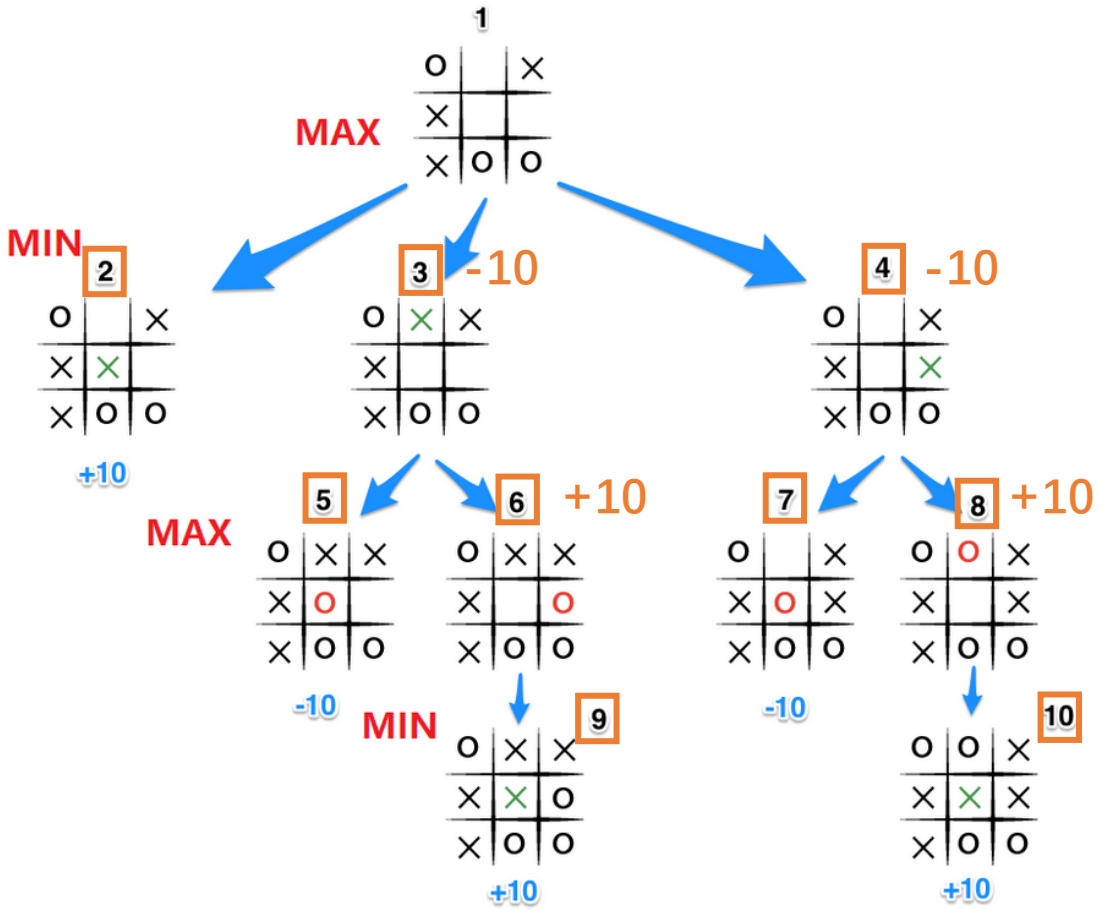
# Minimax搜索



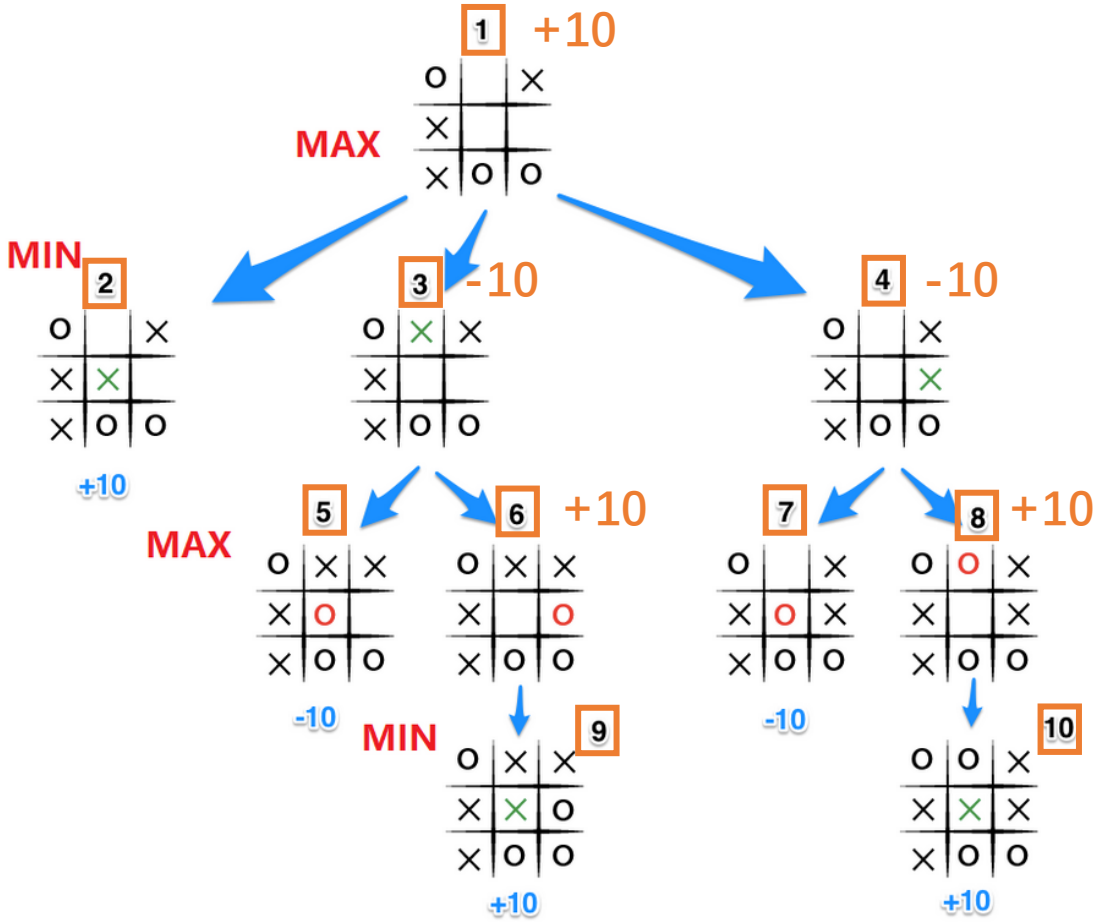
# Minimax搜索



# Minimax搜索



# Minimax搜索



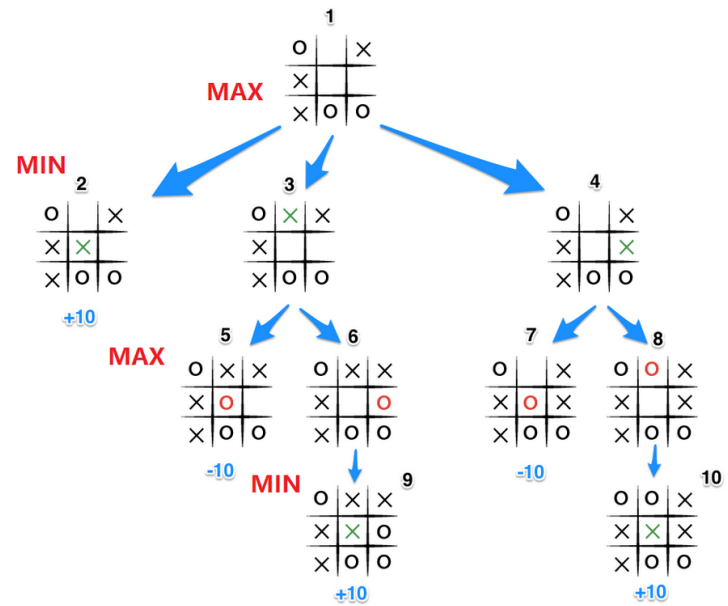

**function** MINIMAX-DECISION($state$) **returns** *an action*
  **return** $\arg\max_{a \in \text{ACTIONS}(s)}$ MIN-VALUE(RESULT($state$, $a$))

---

**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** $v$

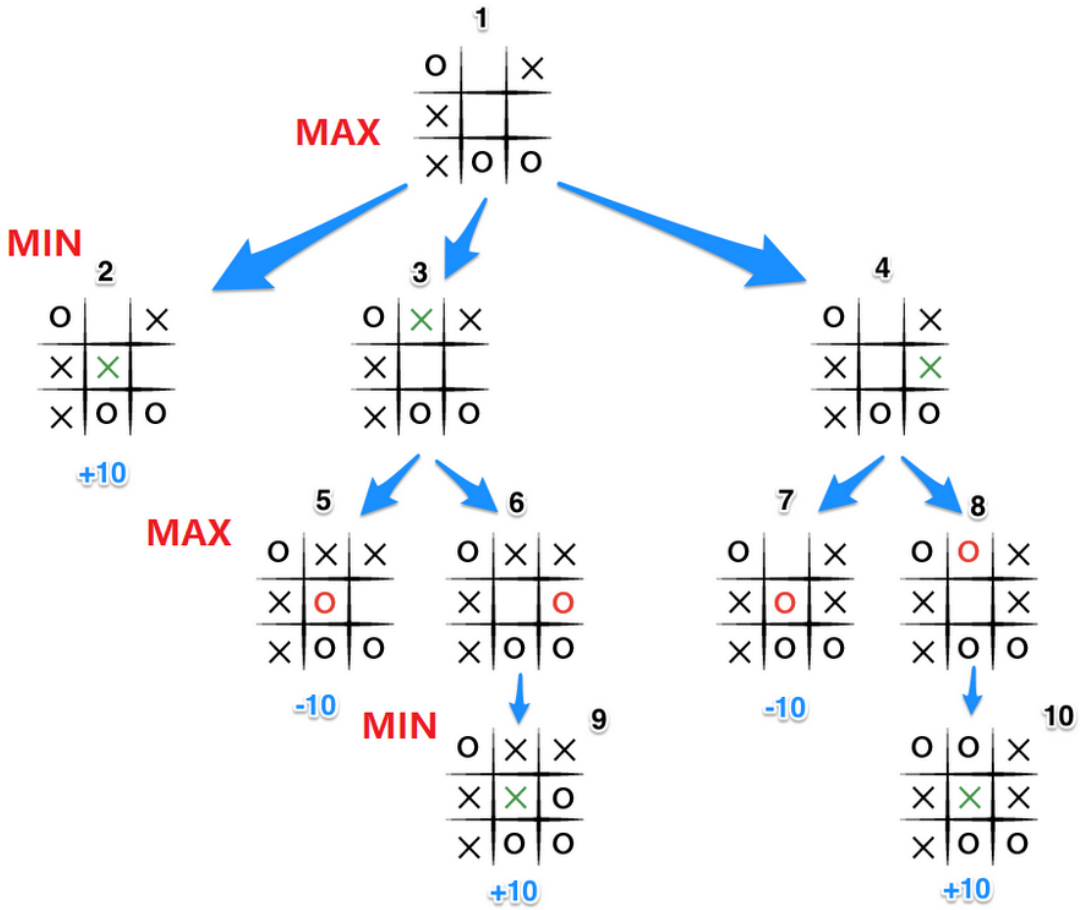


# Alpha-beta剪枝

- 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；
  - 而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。
- Alpha-beta剪枝的一种简单理解：
  - Max节点维护alpha值(已知最大效益值)，Min节点维护beta值(已知最小效益值)
  - Alpha剪枝：Max节点的alpha值 ≥ 其任一祖先Min节点的beta值
  - Beta剪枝：Min节点的beta值 ≤ 其任一祖先Max节点的alpha值

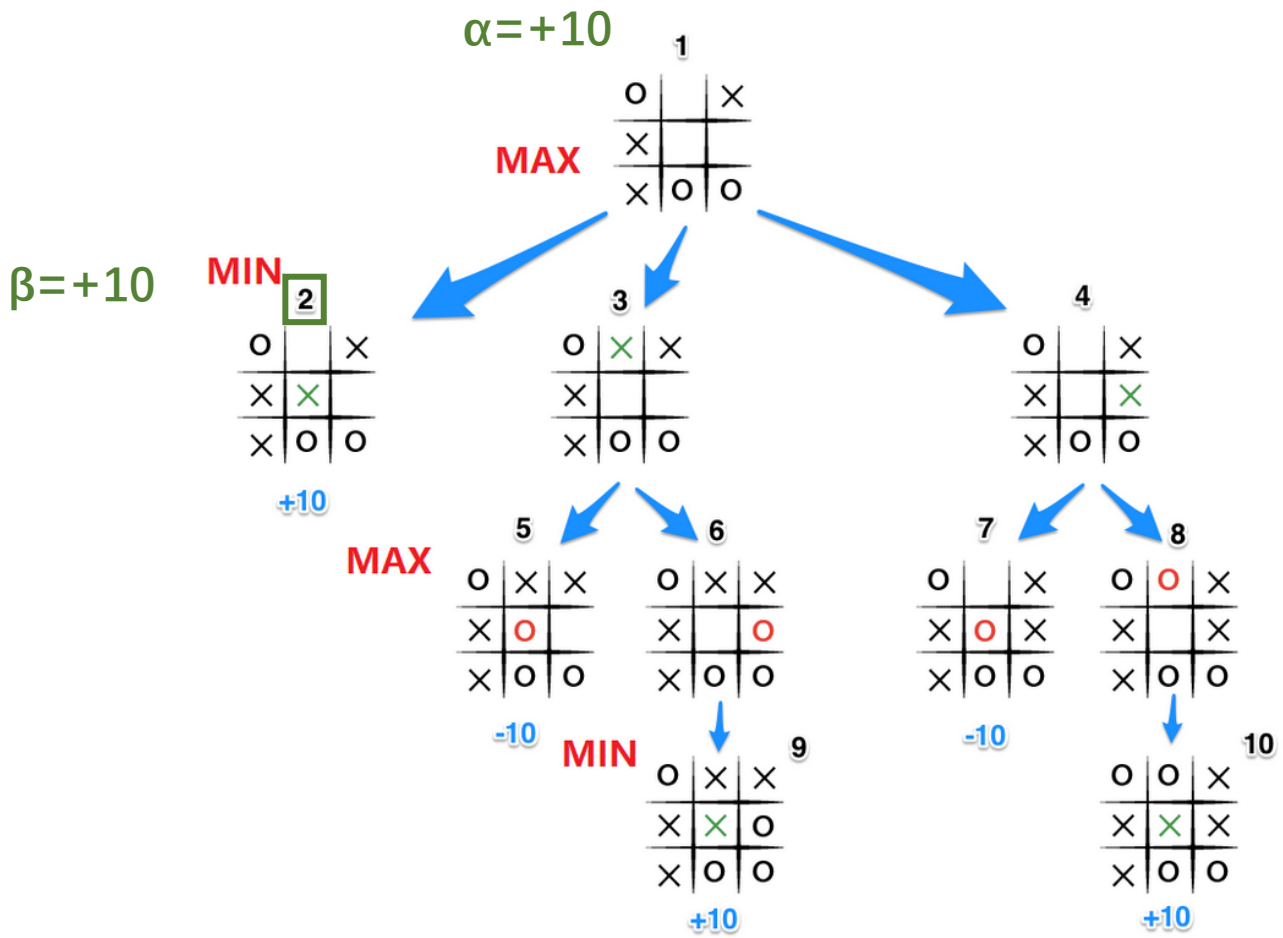
# Alpha-beta剪枝

Max节点的alpha值 ⩾ 其任一祖先Min节点的beta值
Min节点的beta值 ⩽ 其任一祖先Max节点的alpha值



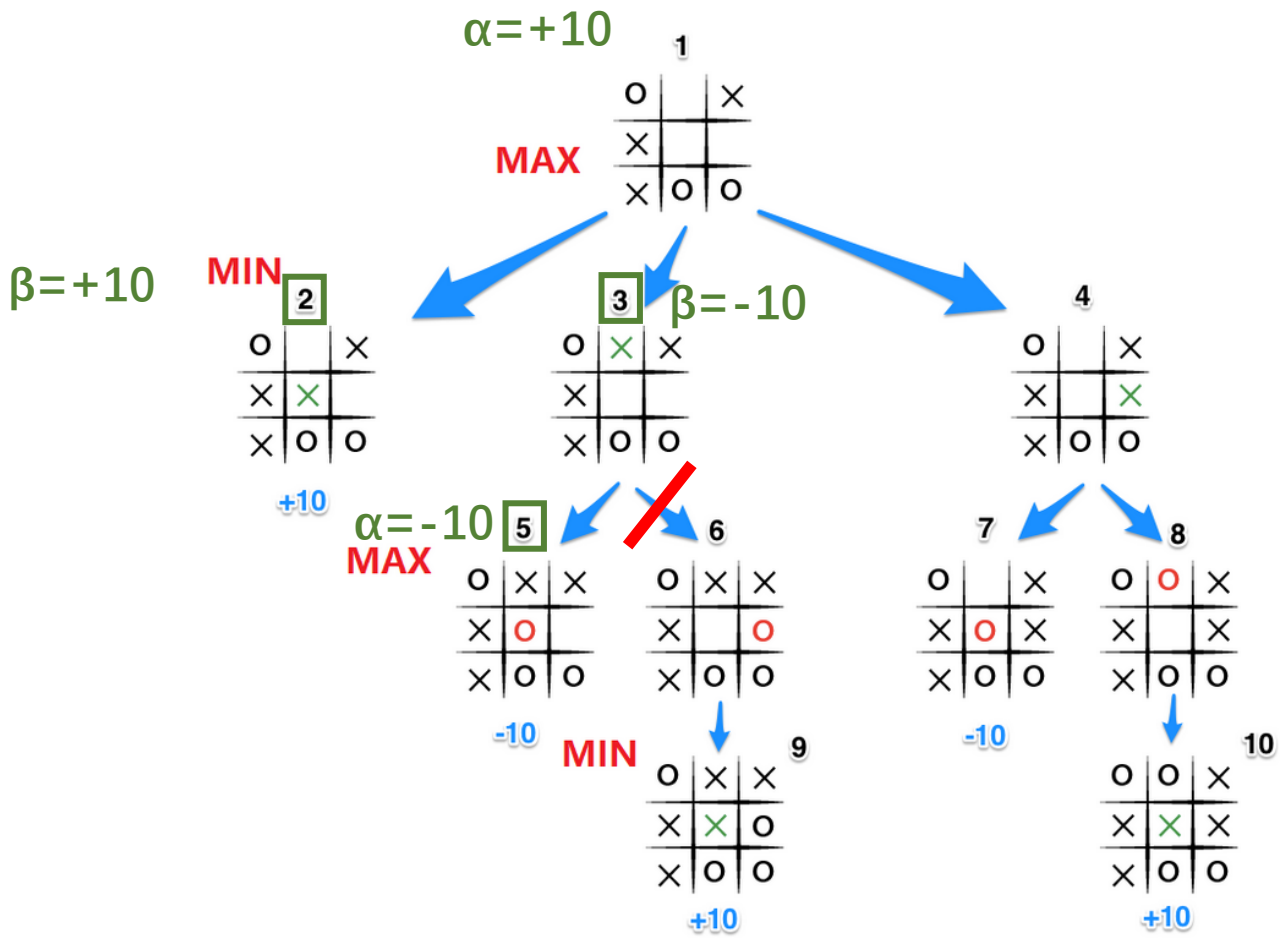
# Alpha-beta剪枝

Max节点的alpha值 ⩾ 其任一祖先Min节点的beta值
Min节点的beta值 ⩽ 其任一祖先Max节点的alpha值



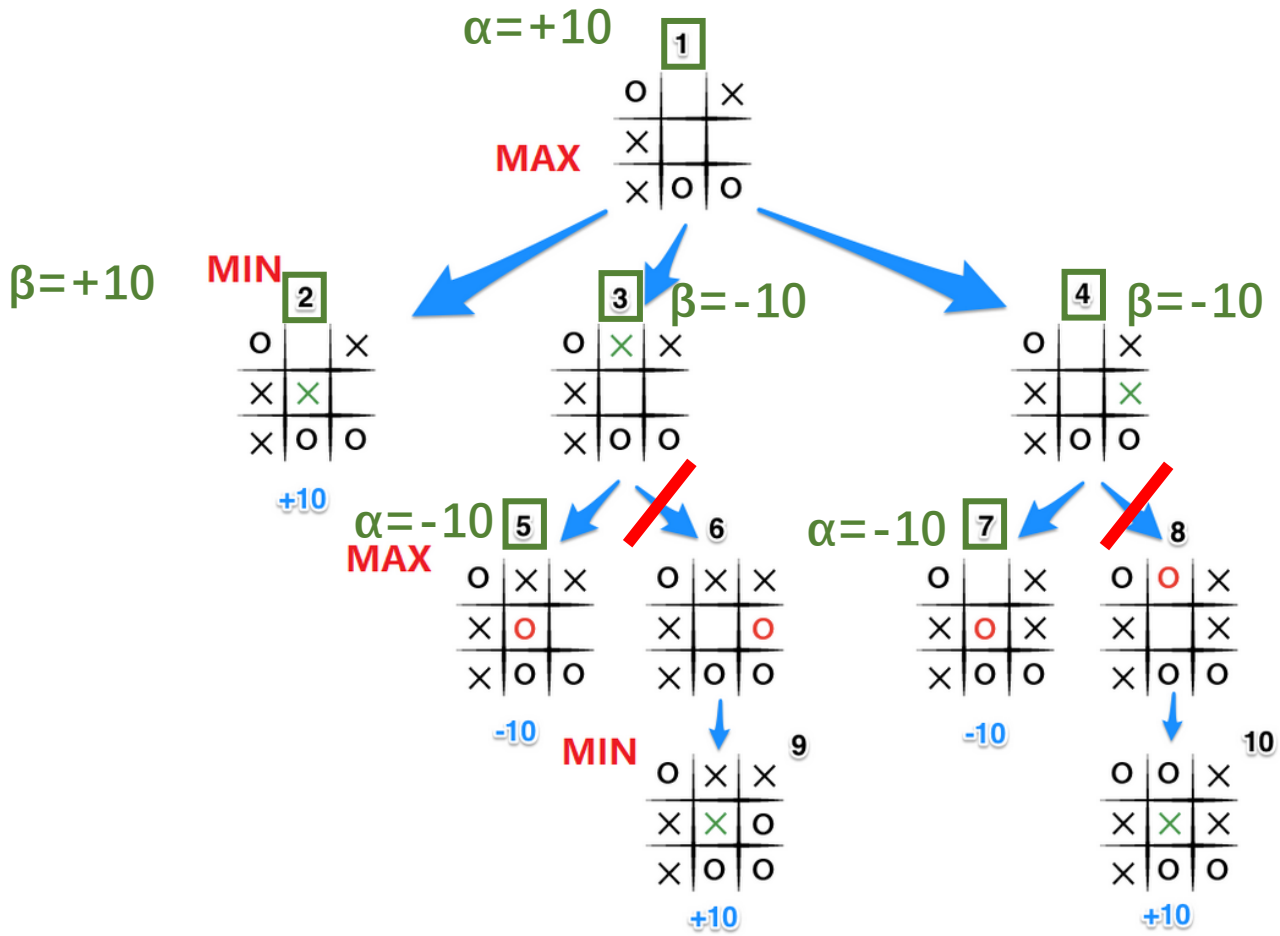
# Alpha-beta剪枝

Max节点的alpha值 ⩾ 其任一祖先Min节点的beta值
Min节点的beta值 ⩽ 其任一祖先Max节点的alpha值



# Alpha-beta剪枝

Max节点的alpha值 ⩾ 其任一祖先Min节点的beta值
Min节点的beta值 ⩽ 其任一祖先Max节点的alpha值



# Alpha-beta剪枝

Max节点的alpha值 ≥ 其任一祖先Min节点的beta值
Min节点的beta值 ≤ 其任一祖先Max节点的alpha值

- 上述过程不利于计算机实现。
  - 如何让每个节点获知祖先Max节点的alpha值，或祖先Min节点的beta值？
- Alpha-beta剪枝算法：
  - 通过函数参数的形式递归传递如下内容：
    - 祖先Max节点中最大的alpha值，以及祖先Min节点中最小的beta值
  - 节点维护“效益值”，Max节点更新上述alpha值，Min节点更新上述beta值
  - Max节点的alpha剪枝：效益值 ≥ 祖先Min节点的最小beta值
  - Min节点的beta剪枝：效益值 ≤ 祖先Max节点的最大alpha值


**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($s$,$a$), $\alpha, \beta$))
    **if** $v \geq \beta$ **then return** $v$
    $\alpha \leftarrow$ MAX($\alpha$, $v$)
  **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($s$,$a$) , $\alpha, \beta$))
    **if** $v \leq \alpha$ **then return** $v$
    $\beta \leftarrow$ MIN($\beta$, $v$)
  **return** $v$

```
function ALPHA-BETA-SEARCH(state) returns an action
    v ← MAX-VALUE(state, −∞, +∞)
    return the action in ACTIONS(state) with value v
```

---

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

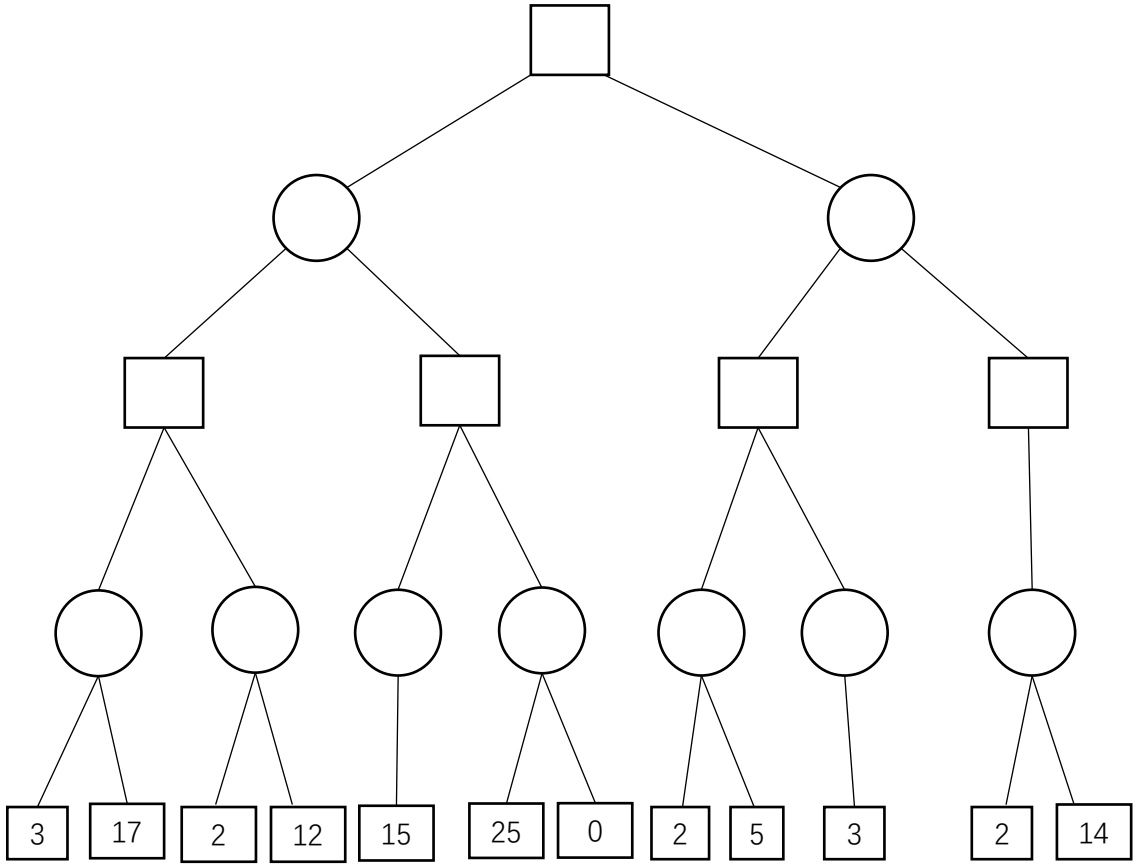
剪枝/更新alpha值

---

```
function MIN-VALUE(state, α, β) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← +∞
    for each a in ACTIONS(state) do
        v ← MIN(v, MAX-VALUE(RESULT(s,a) , α, β))
        if v ≤ α then return v
        β ← MIN(β, v)
    return v
```

剪枝/更新beta值

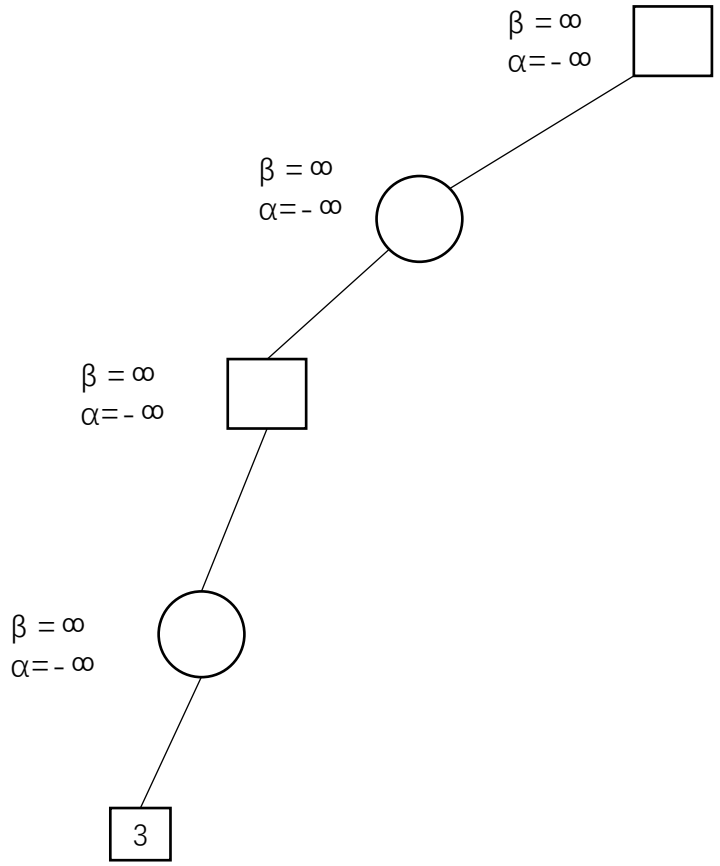
# 练习：Alpha-beta剪枝



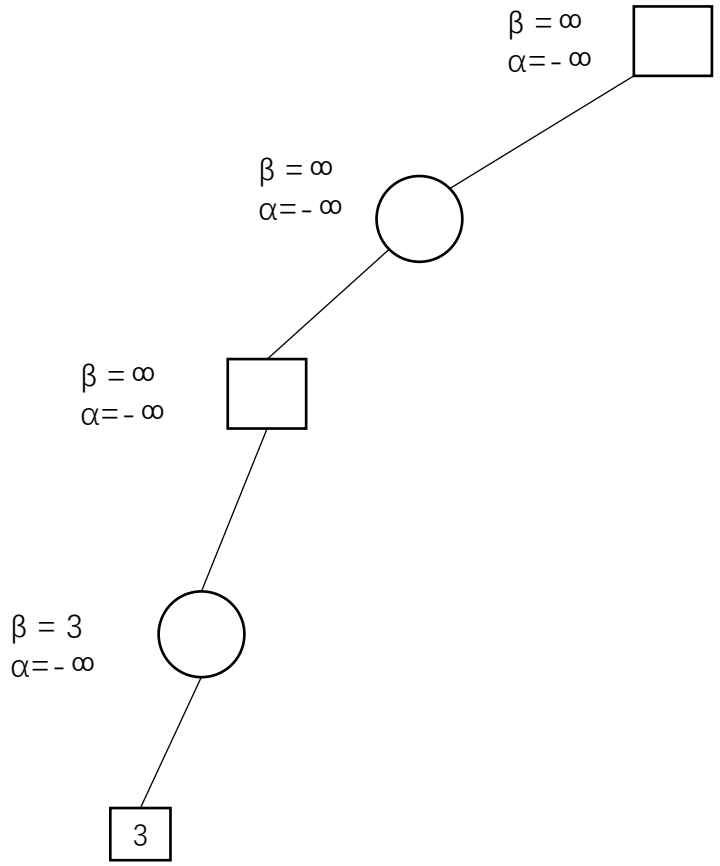

方形：玩家A / Max节点
圆形：玩家B / Min节点

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。

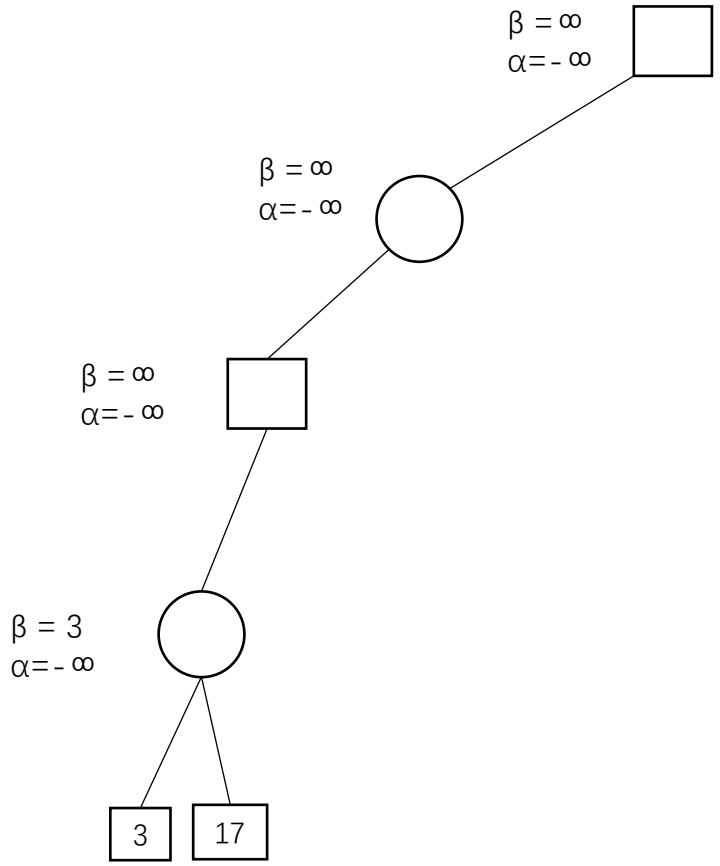
# Alpha-beta剪枝



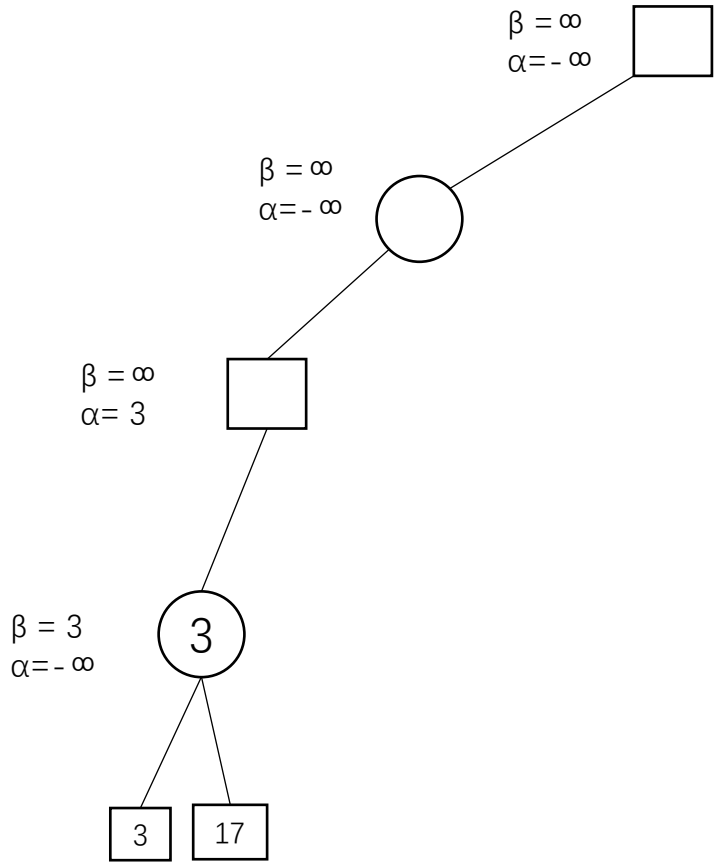
# Alpha-beta剪枝



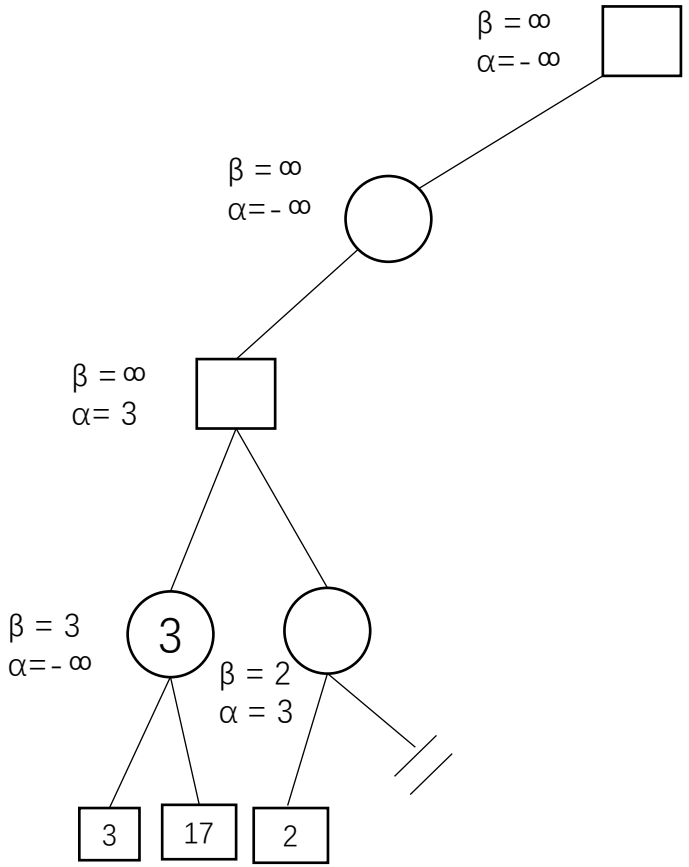
# Alpha-beta剪枝



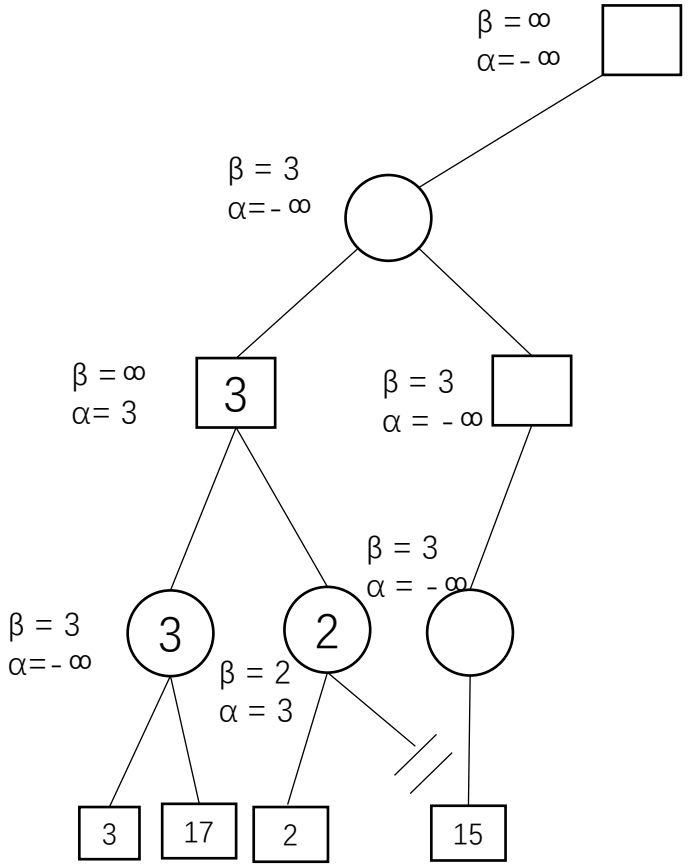
# Alpha-beta剪枝



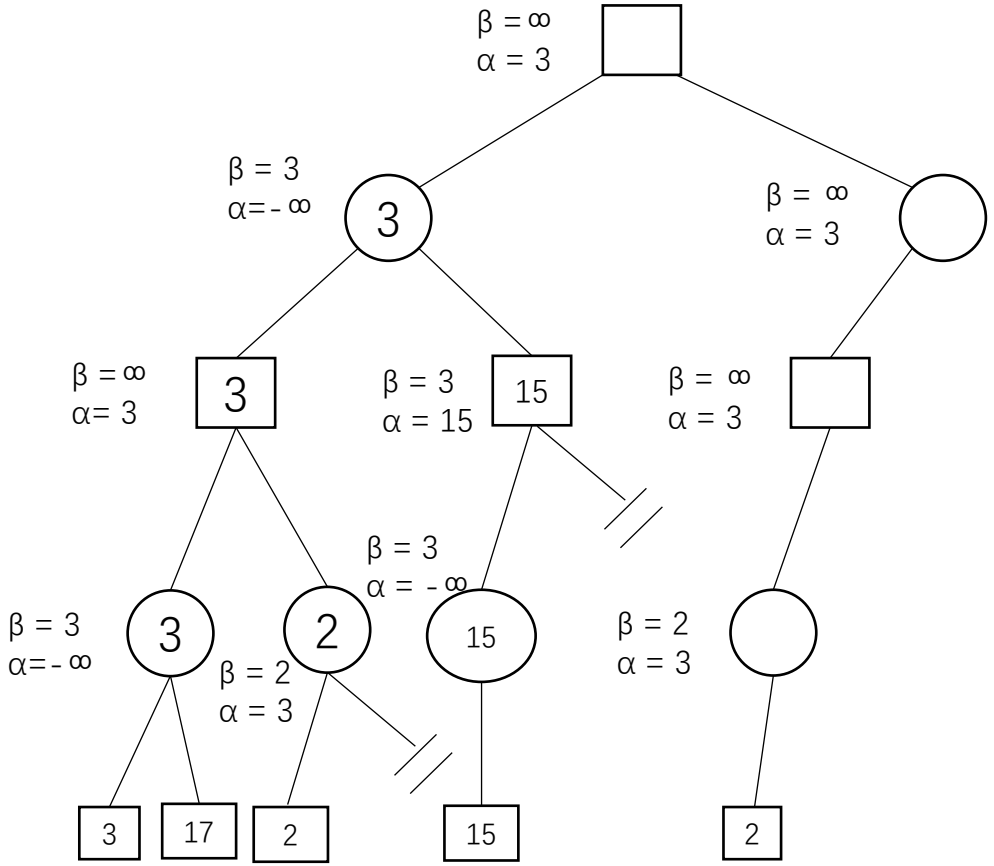
# Alpha-beta剪枝



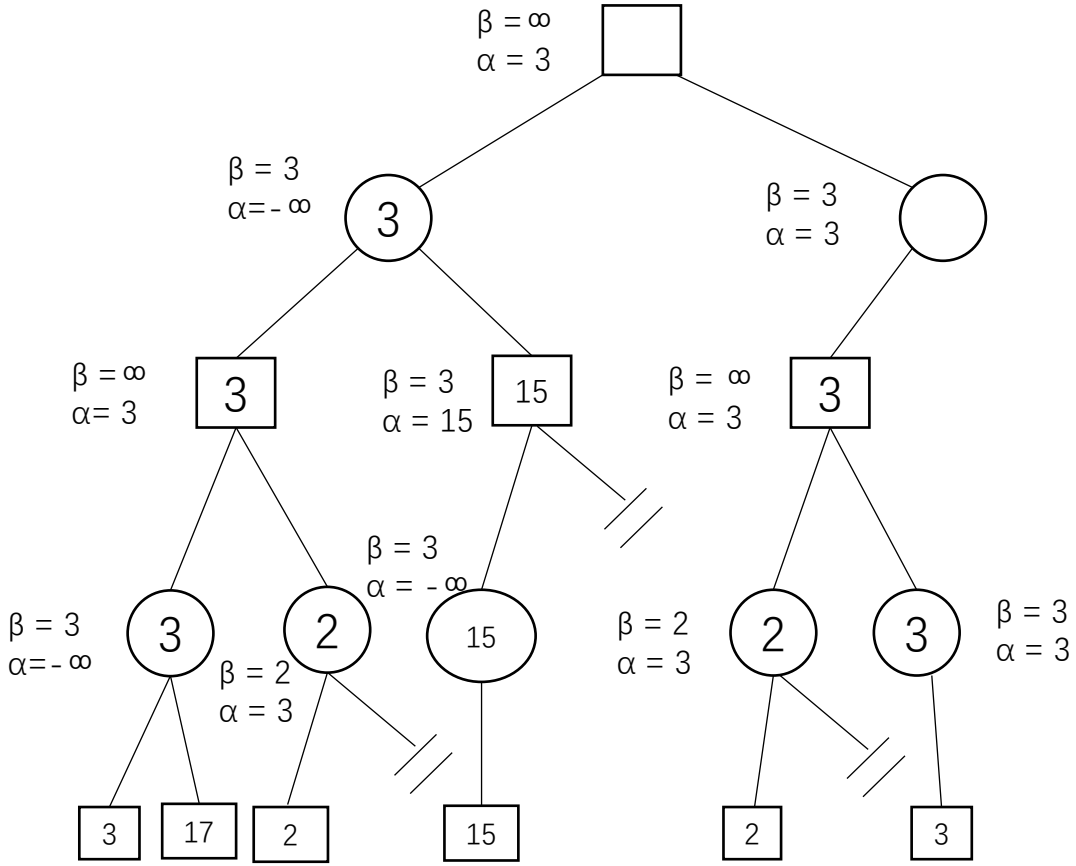
# Alpha-beta剪枝



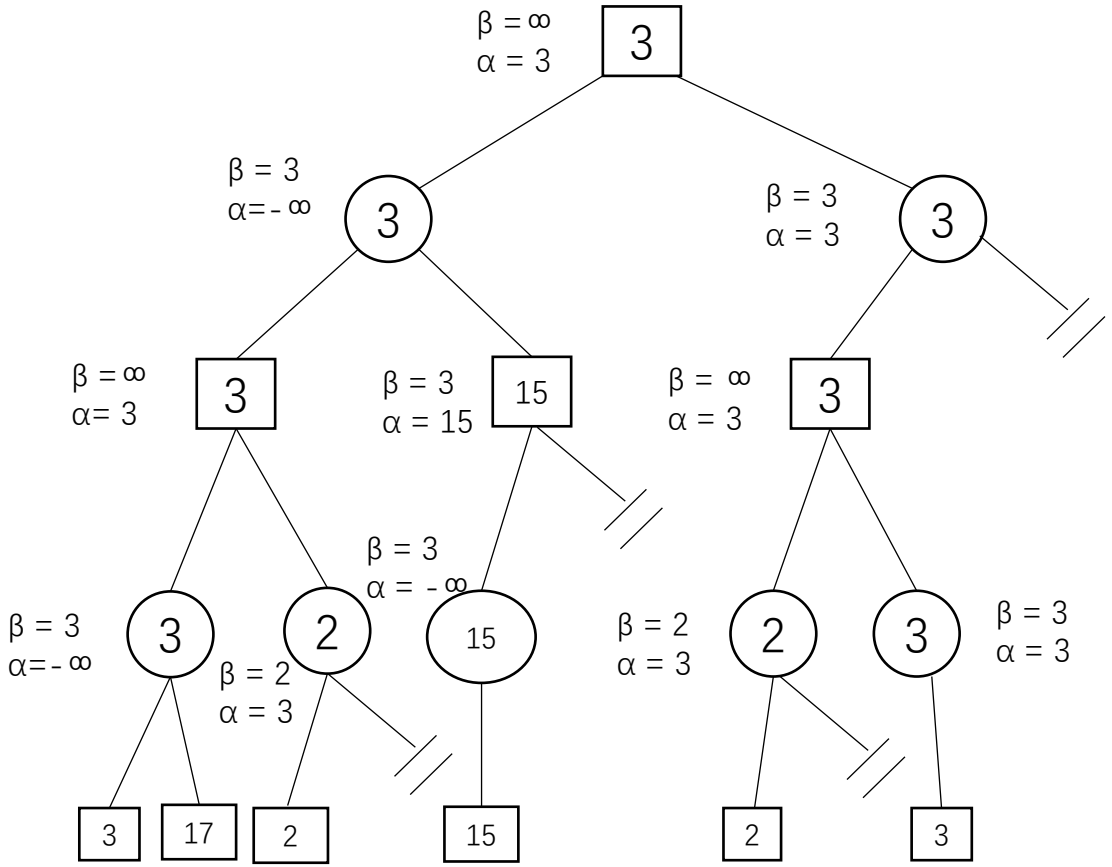
# Alpha-beta剪枝



# Alpha-beta剪枝



# Alpha-beta剪枝



# 评价函数

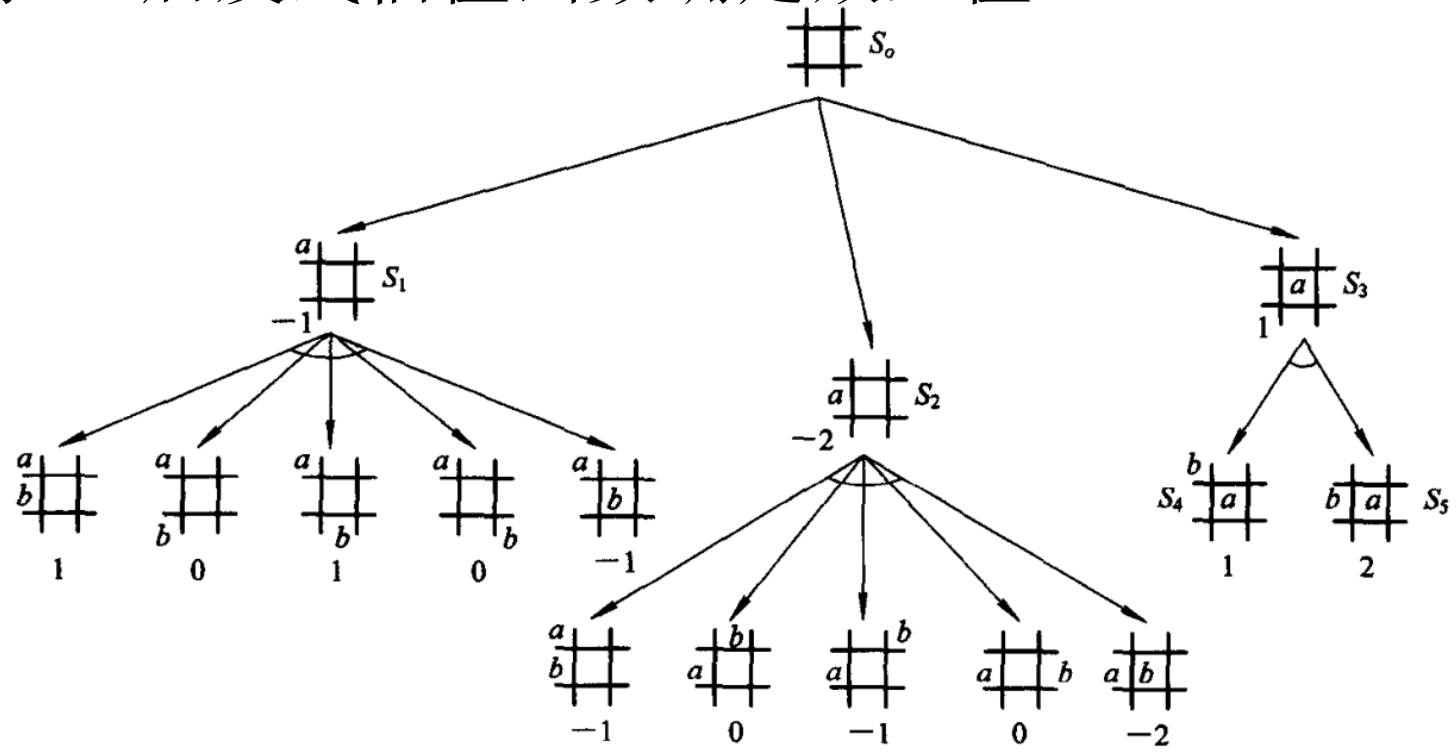
- 在实际问题中，状态数量可能非常庞大。
- 扩展到终止节点的代价是无法容忍的。
- 此时，我们需要限制搜索深度。
- 在有深度限制时，原来的内部节点会充当“叶子节点”的作用。
- 如何得到这些内部节点的“效益值”？

- 我们使用“评价函数”，对节点进行优劣评分。
- 此时以评价函数值作为节点分值的估计。

# 启发式估值函数

- 在实际情况中，构建完整博弈树是不现实的

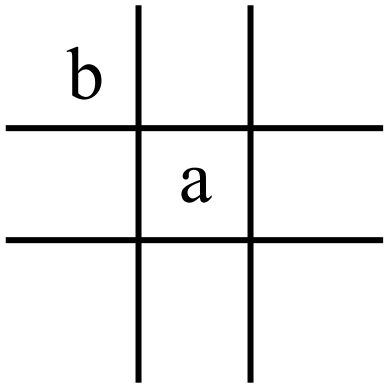
- 扩展一定深度的博弈树 + 启发式估值函数确定效益值



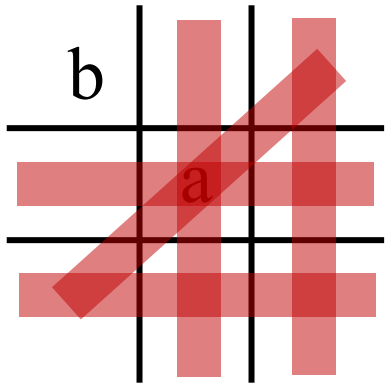
# 启发式估值函数

- 设棋局为P, 估价函数为e(P)
- e(P)=e(+P)-e(-P)
- e(+P)表示棋局P上有可能使a成为三子成一线的数目
- e(-P)表示棋局P上有可能使b成为三子成一线的数目

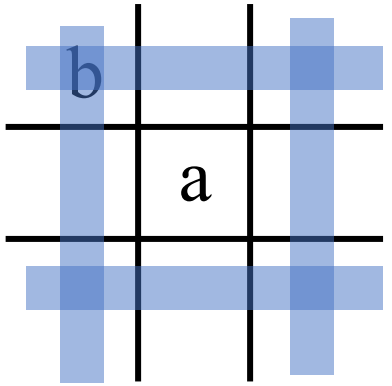
$S_4$




e(P)=5-4=1

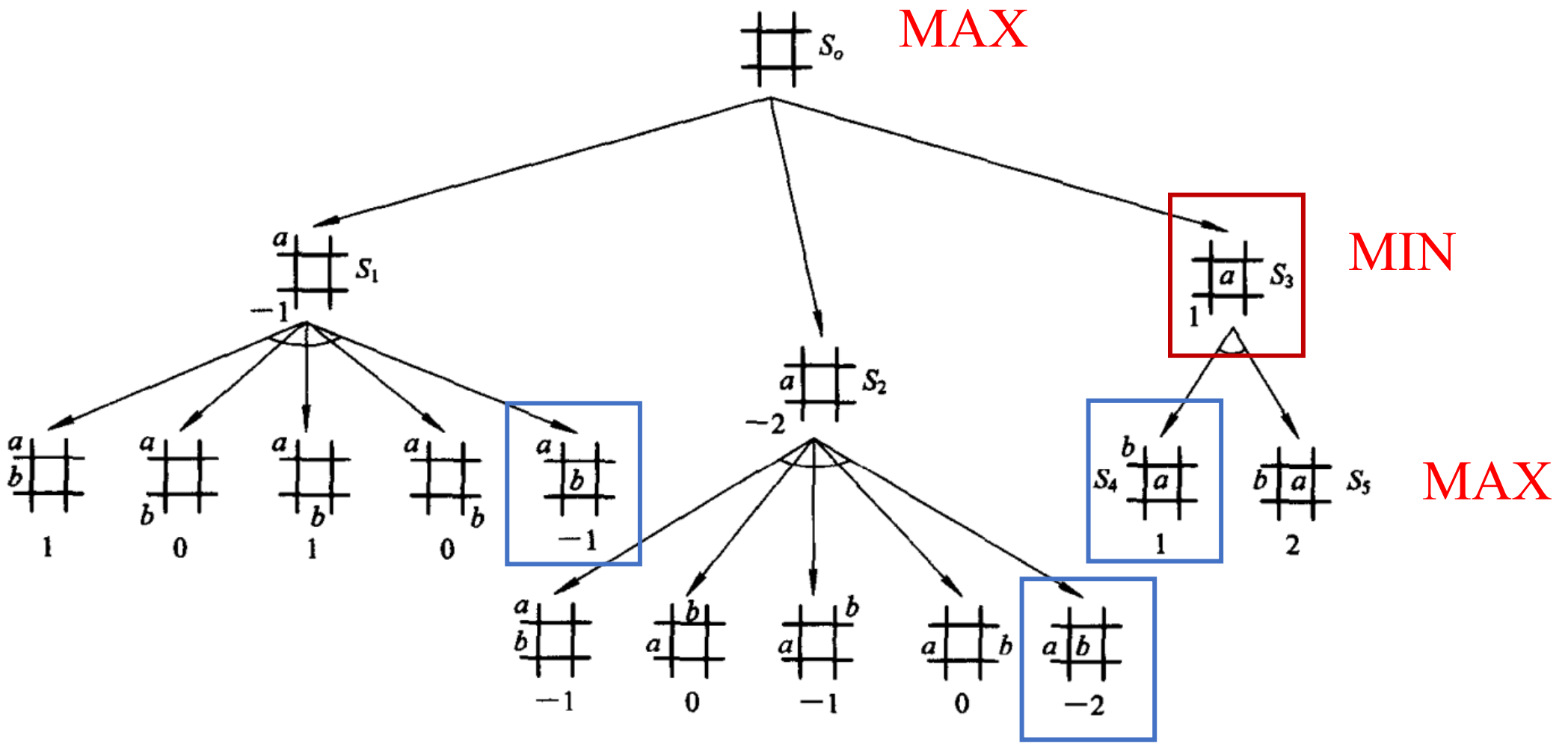



e(+P)=5




e(-P)=4

# 启发式估值函数



# 实验4：安排

- 授课安排

| 周次 | 实验内容 |
| --- | --- |
| 7 | 博弈树搜索 |
| 8 | 高级搜索 |

- 实验安排：黑白棋AI
  - 基础要求：使用Alpha-beta剪枝
  - 进阶要求：结合模拟退火算法或遗传算法，优化评价函数

- 截止时间：2023年4月17日23:59

# 实验任务与报告提交

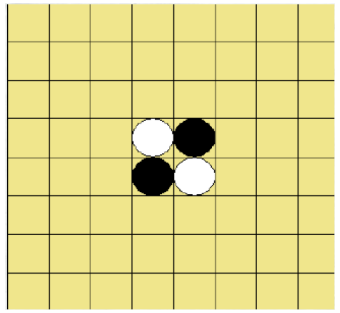
- 实现8×8的黑白翻转棋的人机对战：
  - 要求使用alpha-beta剪枝；
  - 不要求实现UI；
  - 搜索深度和评价函数不限，自己设计。在报告中说明清楚自己的评价函数及搜索策略。
  - 鼓励大家结合高级搜索算法优化评价函数。
  - 实验结果要求展示至少连续三个回合（人和机器各落子一次指一回合）的棋局分布情况，并输出每步落子的得分。
- 提交文件：20******_wangxiaoming.zip，内含

① 实验报告：20******_wangxiaoming.pdf

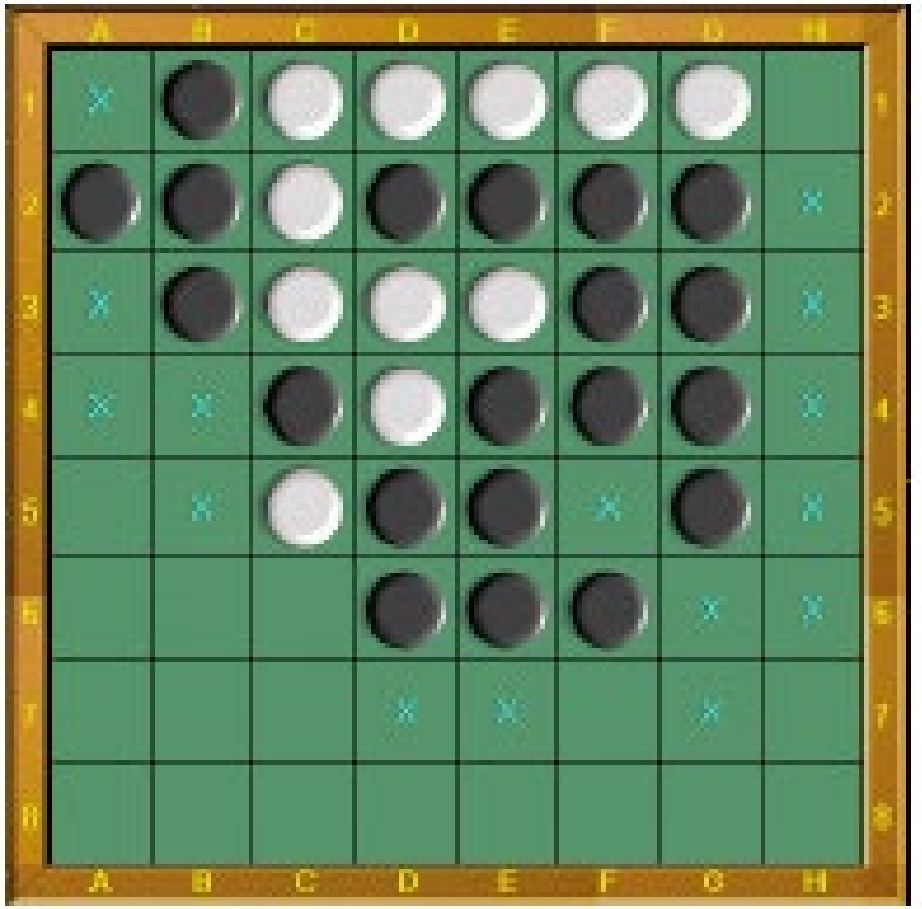
② 代码：/code　　如果代码分成多个文件，需要写readme

# 黑白棋

- 黑棋先手，初始棋局如下：



- 只允许在可以翻转的位置落子
- 落子后，如果在横排、竖排、对角线上有另一自己颜色的棋子，则夹在中间连续排布的另一颜色的棋子会翻转



https://baike.baidu.com/item/%E9%BB%91%E7%99%BD%E6%A3%8B/80689