BUSS 3620.人工智能导论

搜索 II

刘佳璐

安泰经济与管理学院

上海交通大学

BUSS 3620.人工智能导论

#1. 涉及到多智能体的搜索

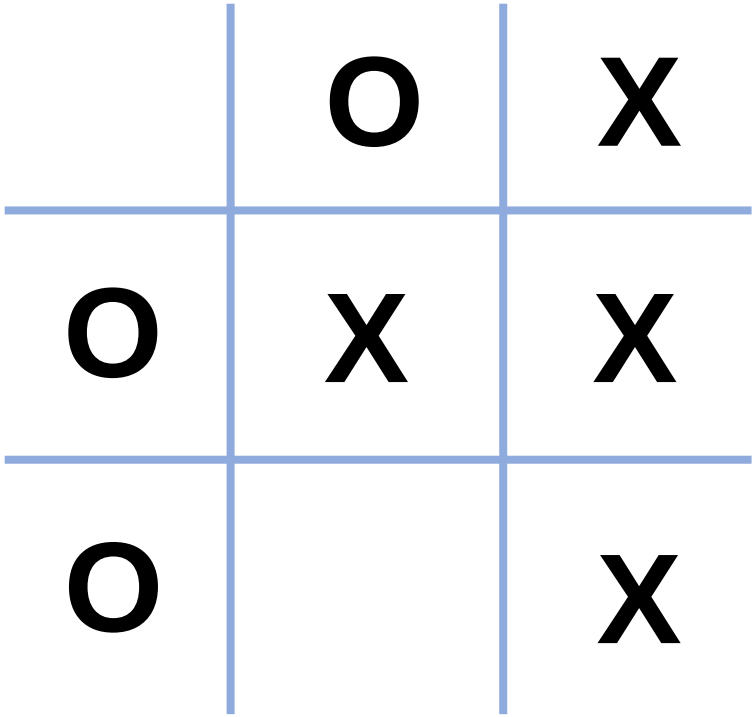
刘佳璐

安泰经济与管理学院

上海交通大学

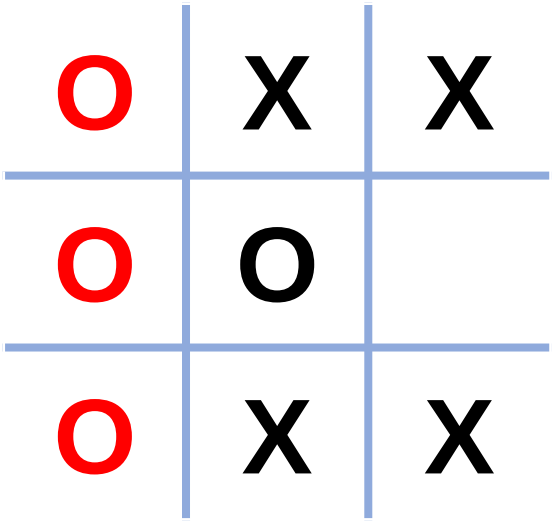
对抗搜索 Adversarial Search

- 有一个**对手**：目标与智能体相反
  - 游戏

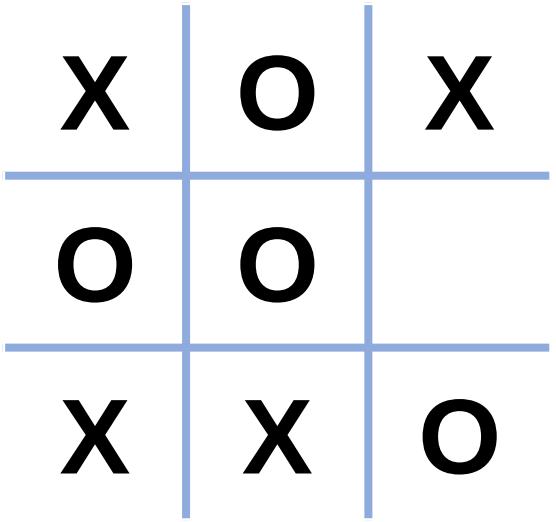


极小极大算法 MiniMax Algorithm

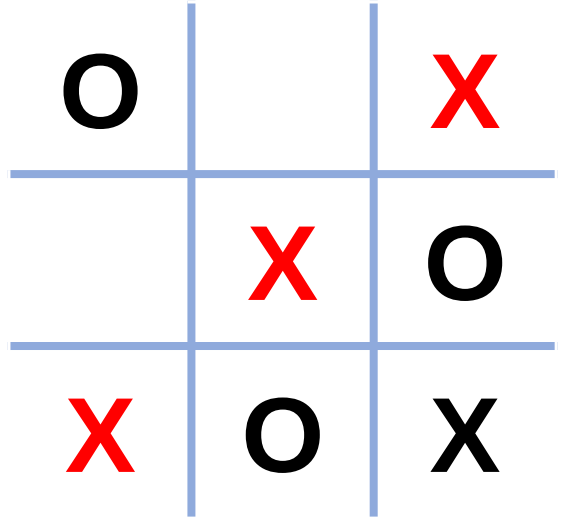
- 将输赢转换为数字
  - 计算机能够更容易理解



-1



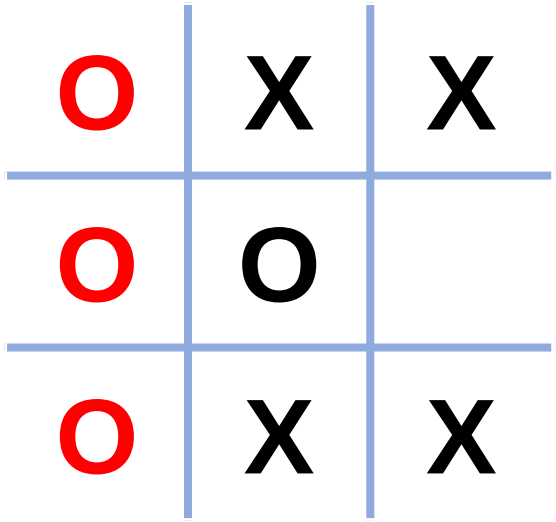
0



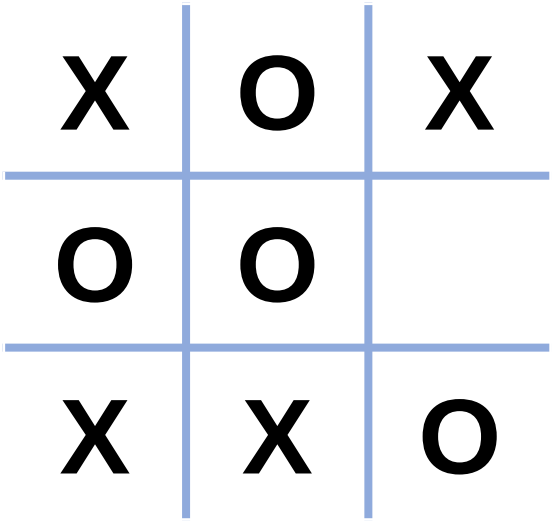
1

极小极大算法 MiniMax Algorithm

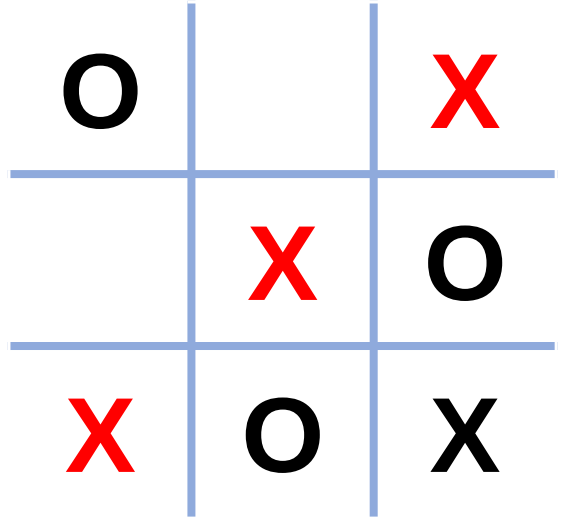
- X选手: Max选手 –目的是**最大化**分数
- O选手: Min选手 –目的是**最小化**分数



-1



0



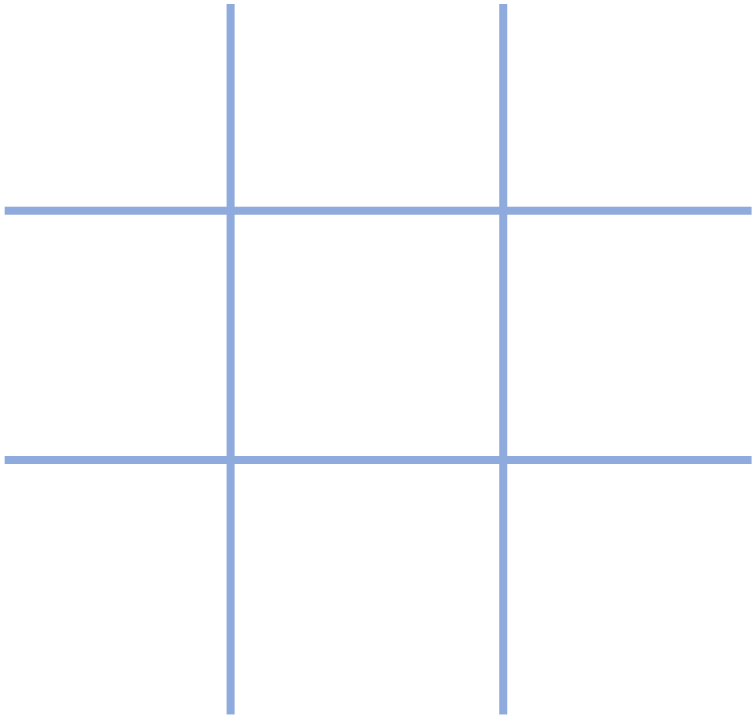
1

游戏 Game

- S_0:初始状态
- Player(s): 返回状态s下能够行动的选手
- Actions(s): 返回状态s的所有合法行动
- Result(s, a): 返回状态s下执行行动a后到达的新状态
- Terminal(s): 确认状态s是否为结束状态(Terminal state)
- Utility(s): 结束状态(terminal state) s的最终数值表示

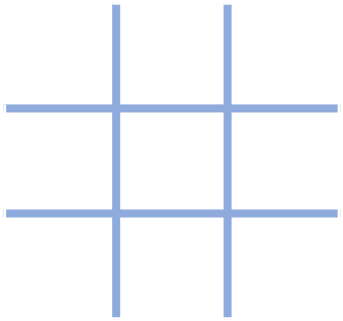
初始状态 Initial State

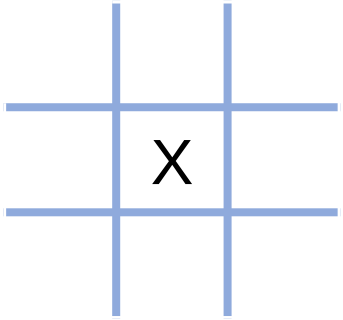
- S_0: 初始状态 initial state



Player(*s*)

- Player(*s*): 返回状态s下能够行动的选手
- 假设X选手是先手

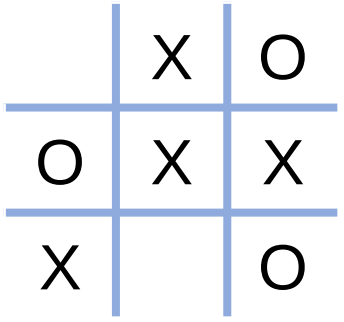
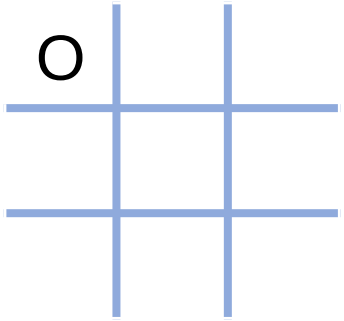
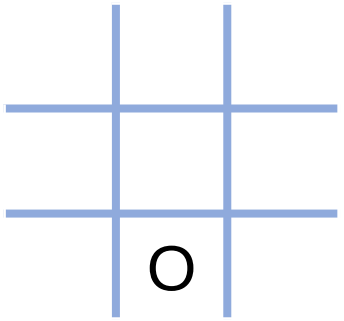
Player() = X

Player(

) = O

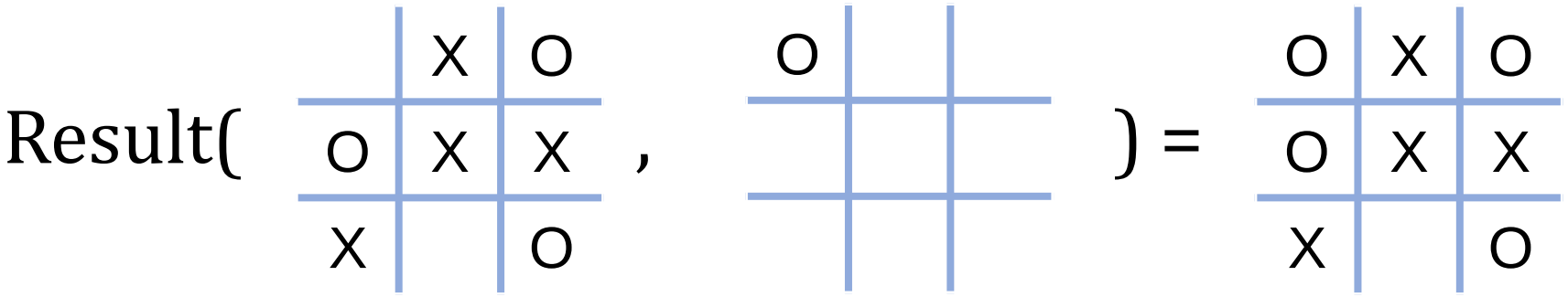
Actions(*s*)

- Actions(*s*): 返回状态s的所有合法行动

Actions() = {  ,  }

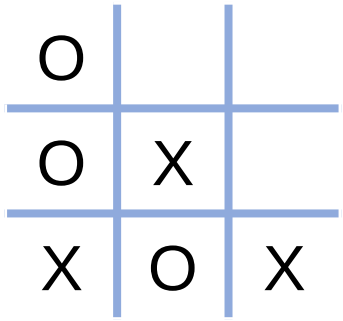
Result(s, a)

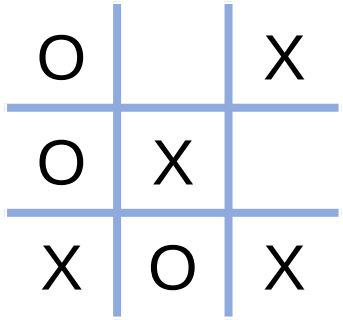
- Result(s, a): 返回状态s下执行行动a后到达的新状态



Terminal(*s*)

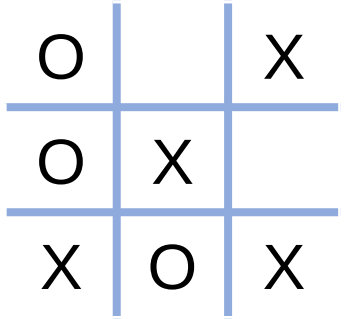
- Terminal(*s*): 确认状态*s*是否为结束状态(Terminal state)

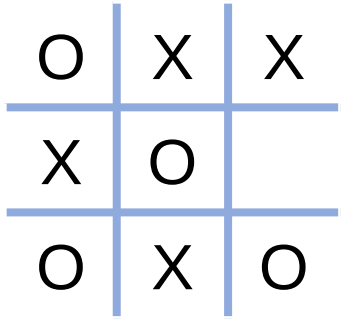
Terminal() = false

Terminal() = true

Utility(s)

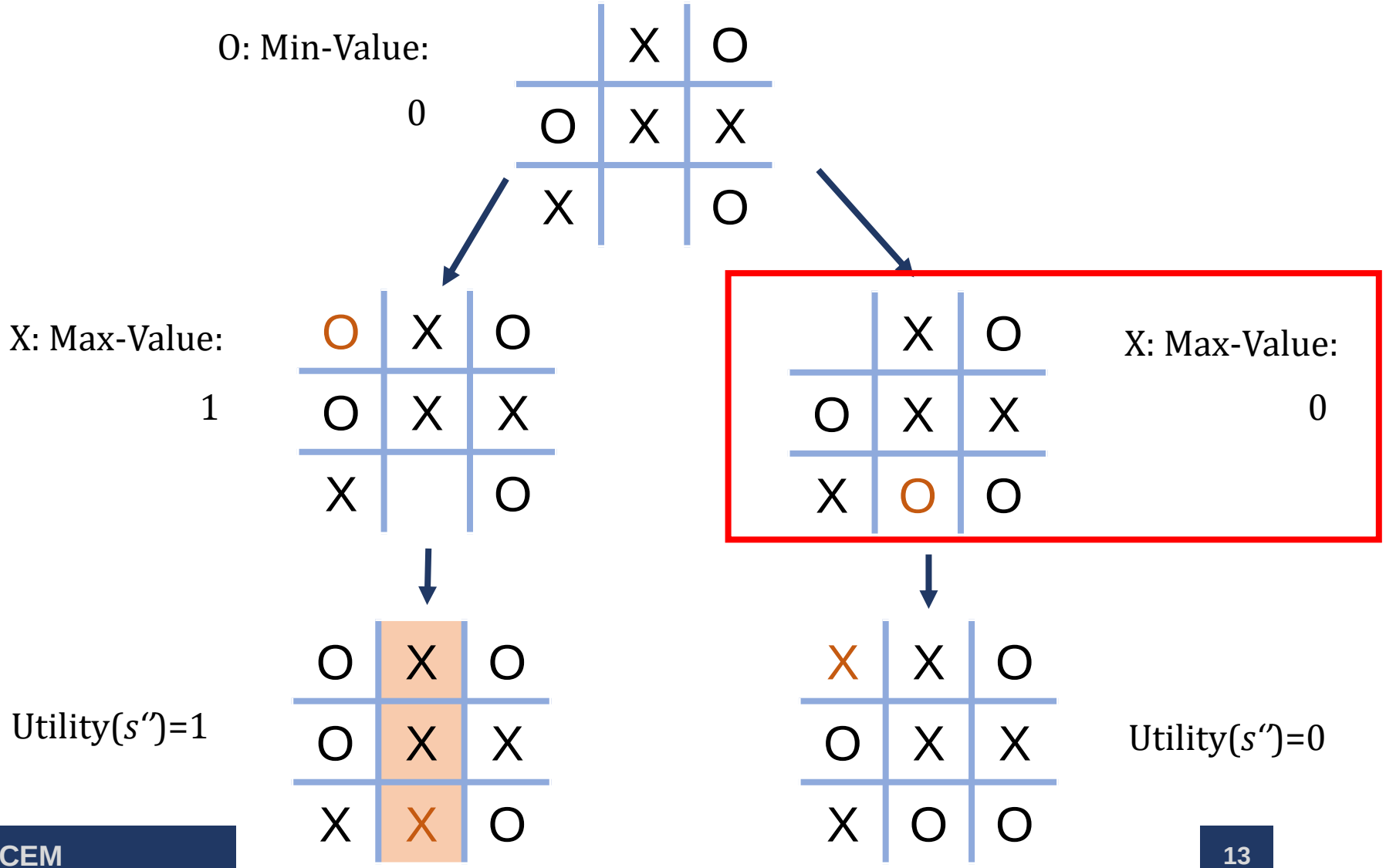
- Utility(s): 结束状态(terminal state) s的最终数值表示

Utility() = 1

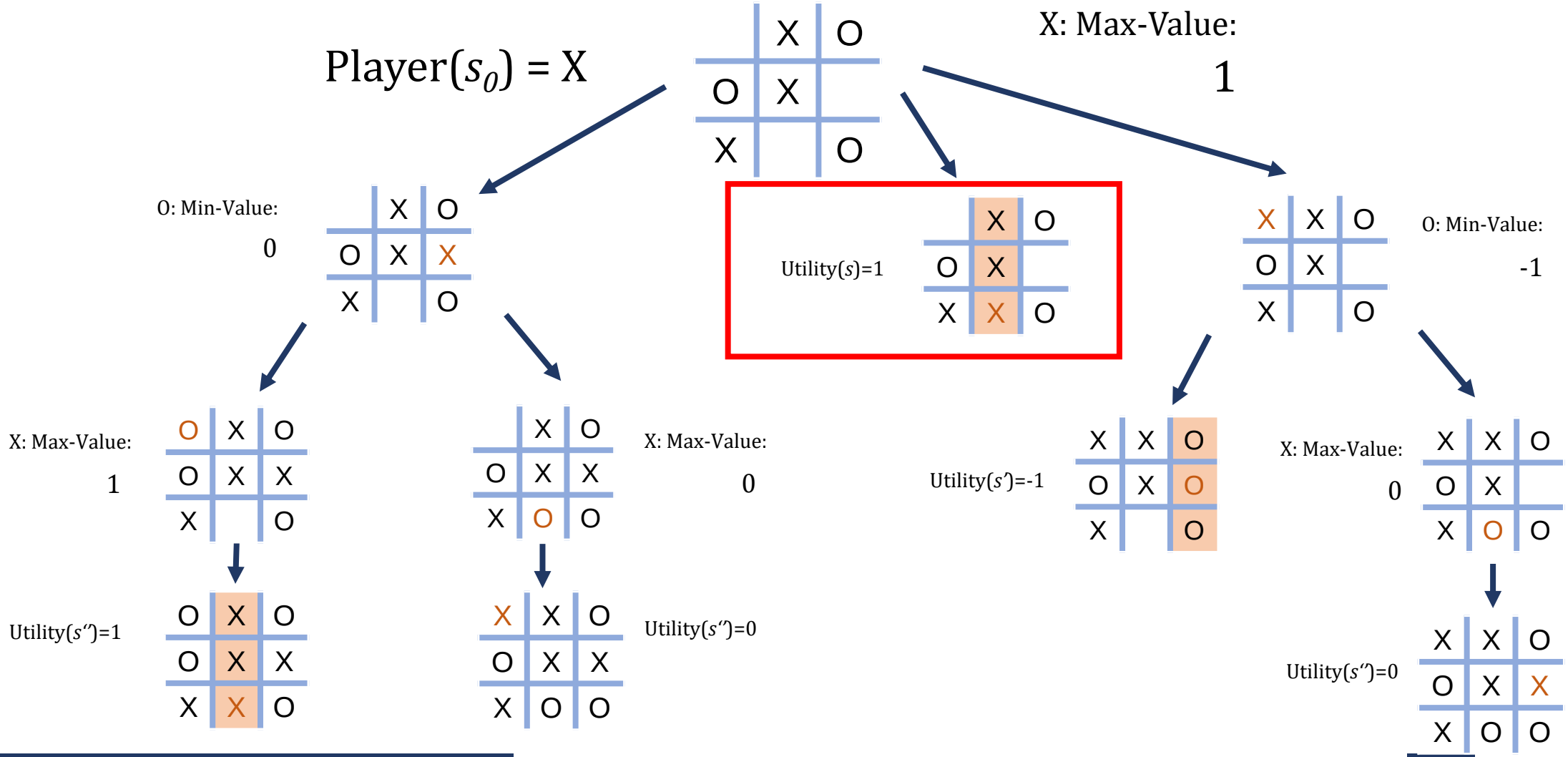
Utility() = -1

如果游戏没有结束，如何表示某个状态的数值?

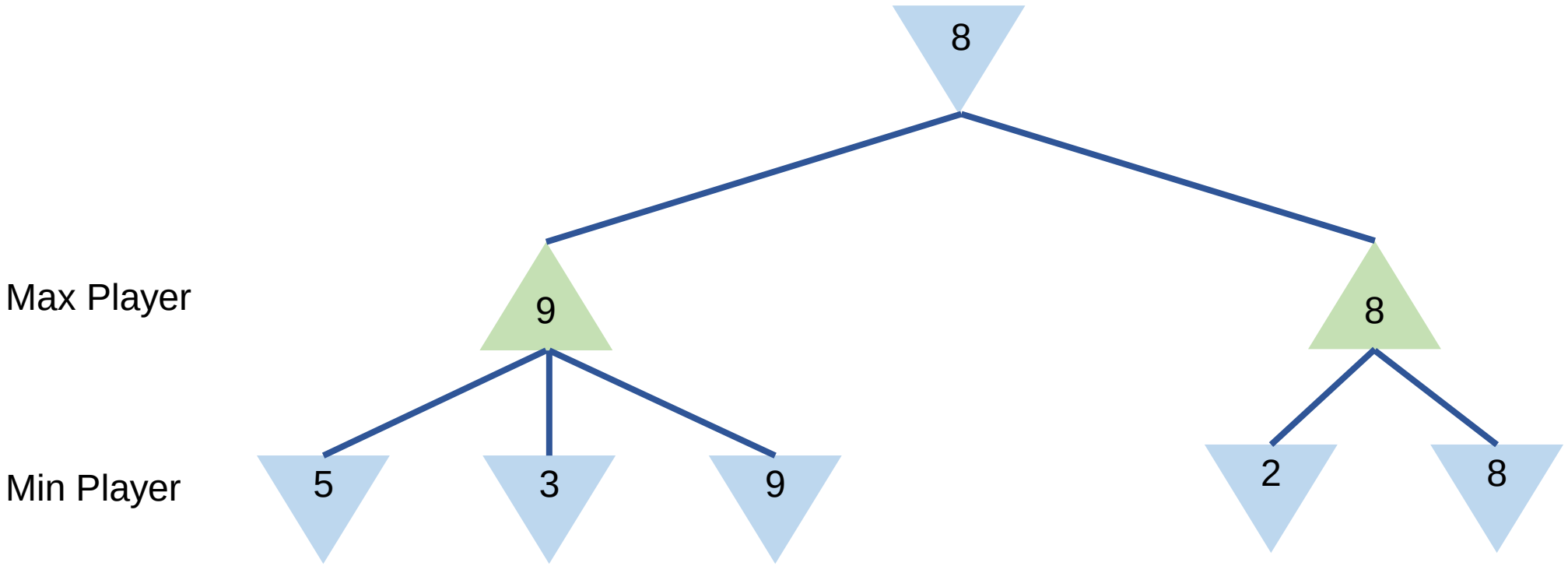
- Player(s) = O
- Player(s') = X



如果游戏没有结束，如何表示某个状态的数值?

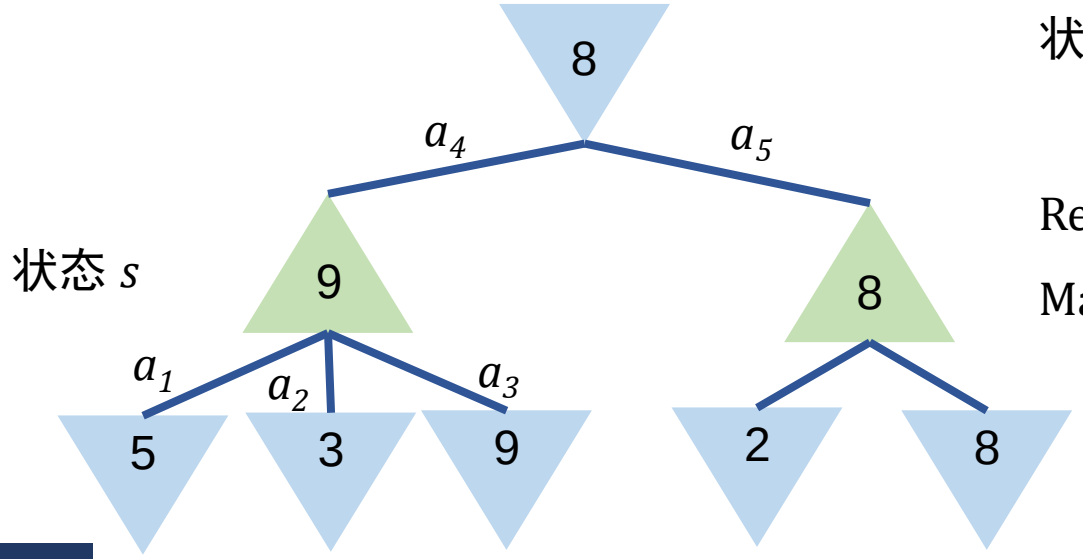


简化成博弈树 game tree



极小极大算法 Minimax

- 给定一个状态 s:
  - Max Player选择Actions(s)中Min-Value(Result(s, a))最大的一个行动a
  - Min Player选择Actions(s)中Max-Value(Result(s, a))最小的一个行动a



状态 s

状态 s'

Result(s', a_5)= s''

Max-Value(Result(s', a_5))= 8

Result(s, a_1)= s_1

Min-Value(Result(s, a_1))= 5

极小极大算法 Minimax

function Max-Value(*state*):

    if Terminal(*state*) is true:

        return Utility(*state*)

    $v=-\infty$

    for *action* in Actions(*state*):

        v = max(v, Min-Value(Result(*state*, *action*)))

    return v

极小极大算法 Minimax

function Min-Value(*state*):

    if Terminal(*state*) is true:

        return Utility(*state*)

    $v = \infty$

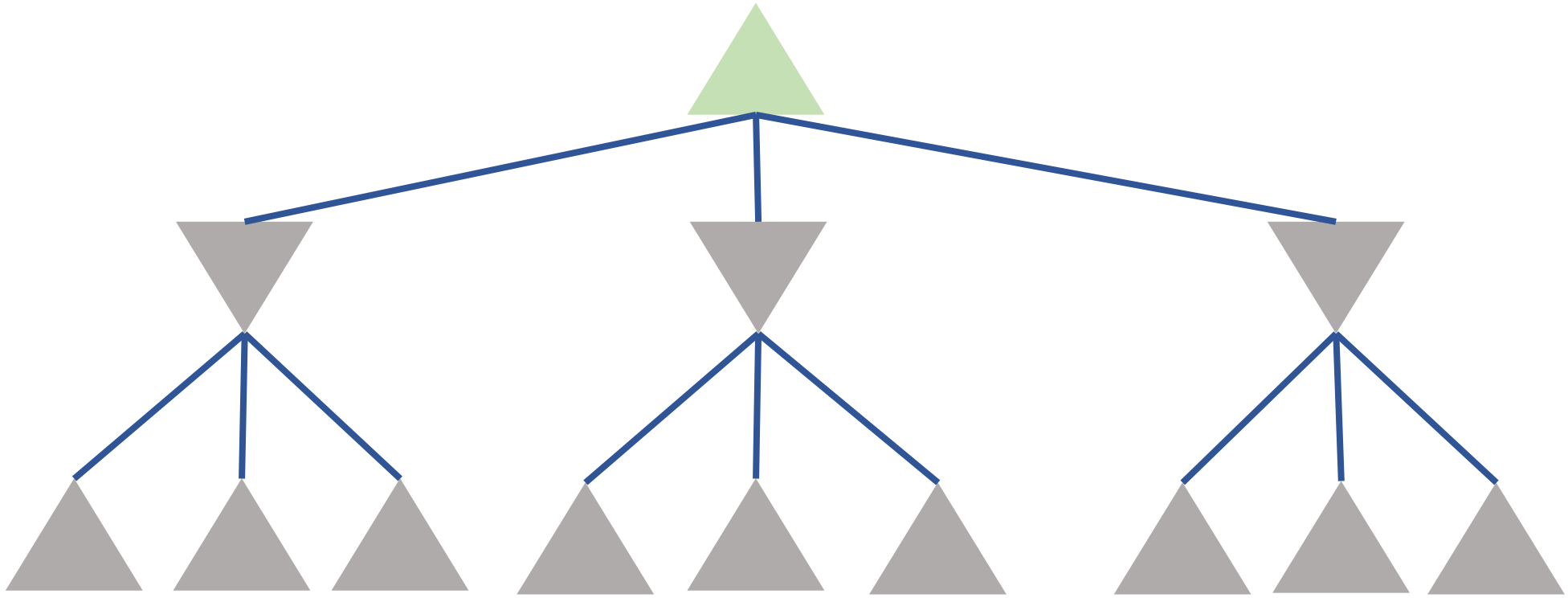
    for *action* in Actions(*state*):

        v = min(v, Max-Value(Result(*state*, *action*)))

    return v

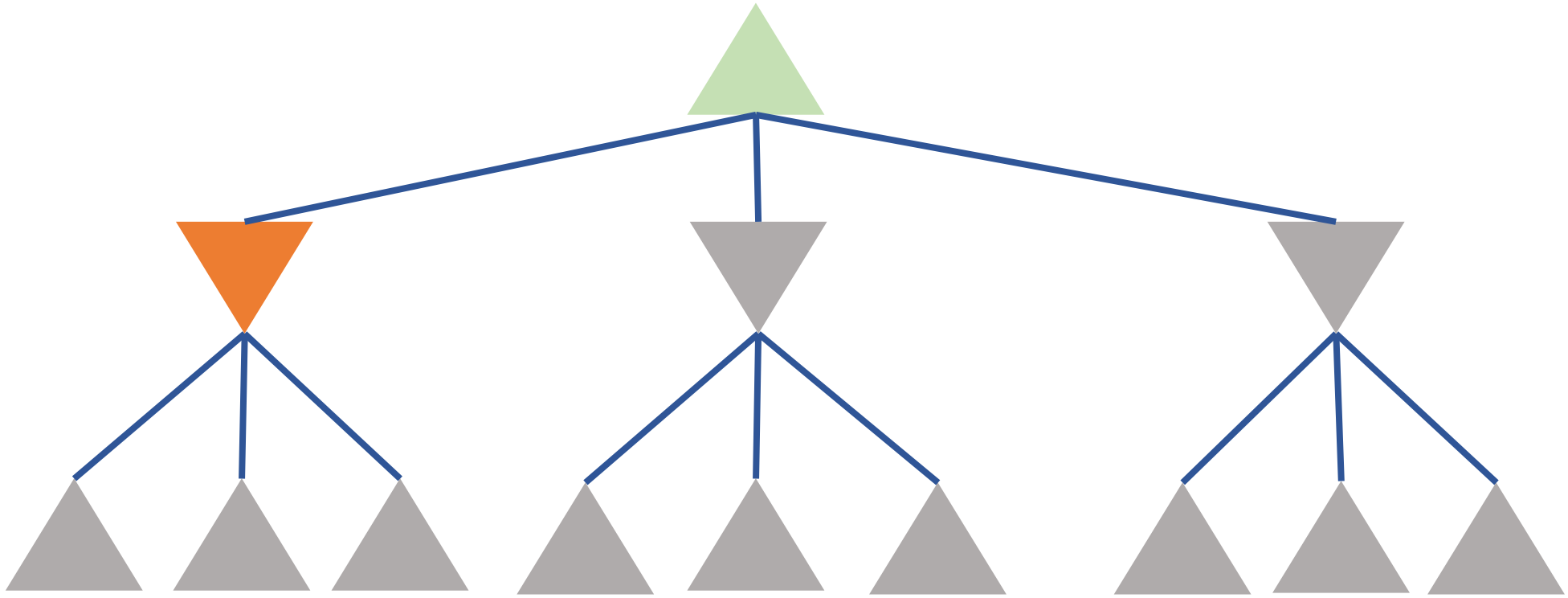
极小极大算法 Minimax

- 两个选手
- 两次行动



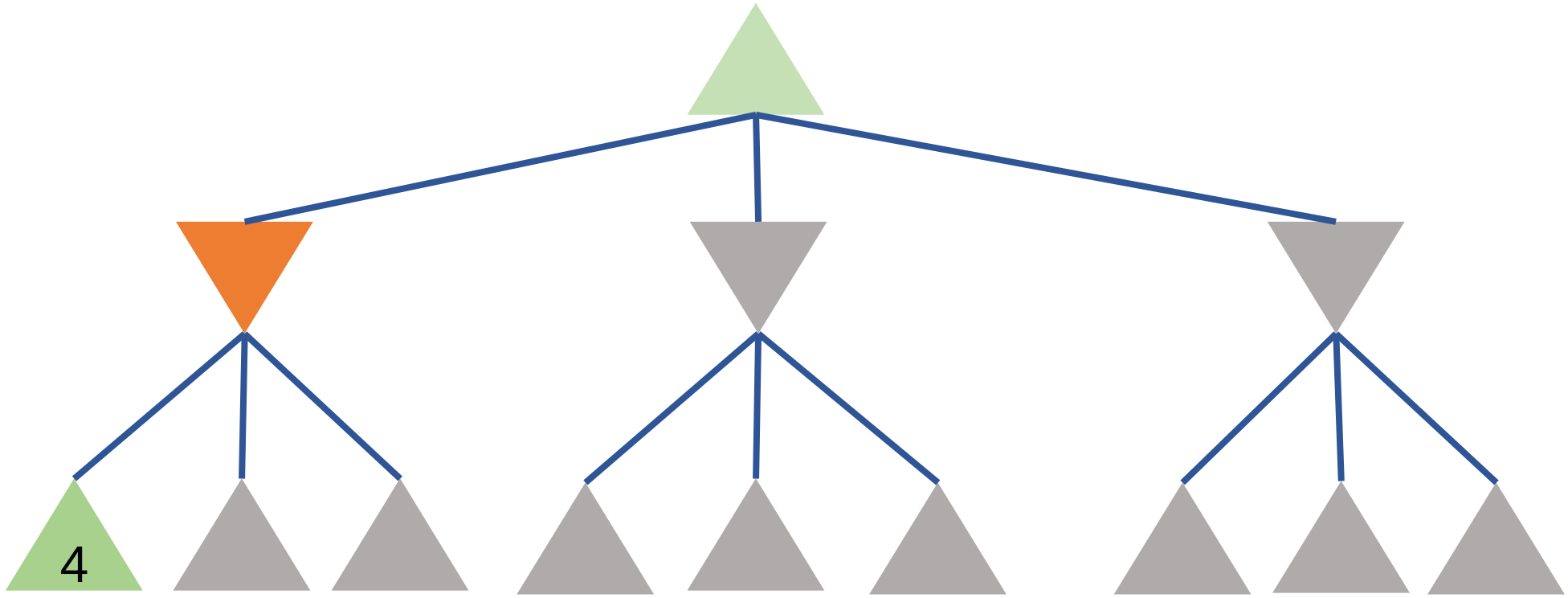
极小极大算法 Minimax

- 两个选手
- 两次行动



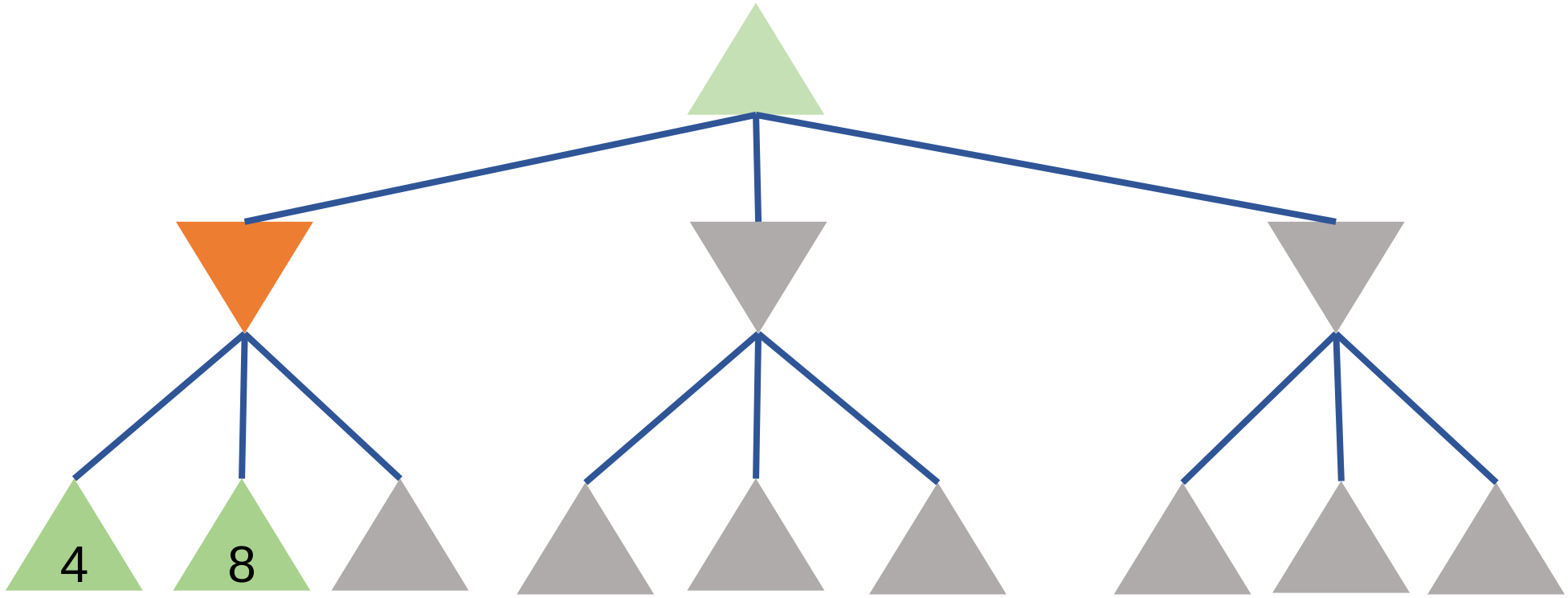
极小极大算法 Minimax

- 两个选手
- 两次行动



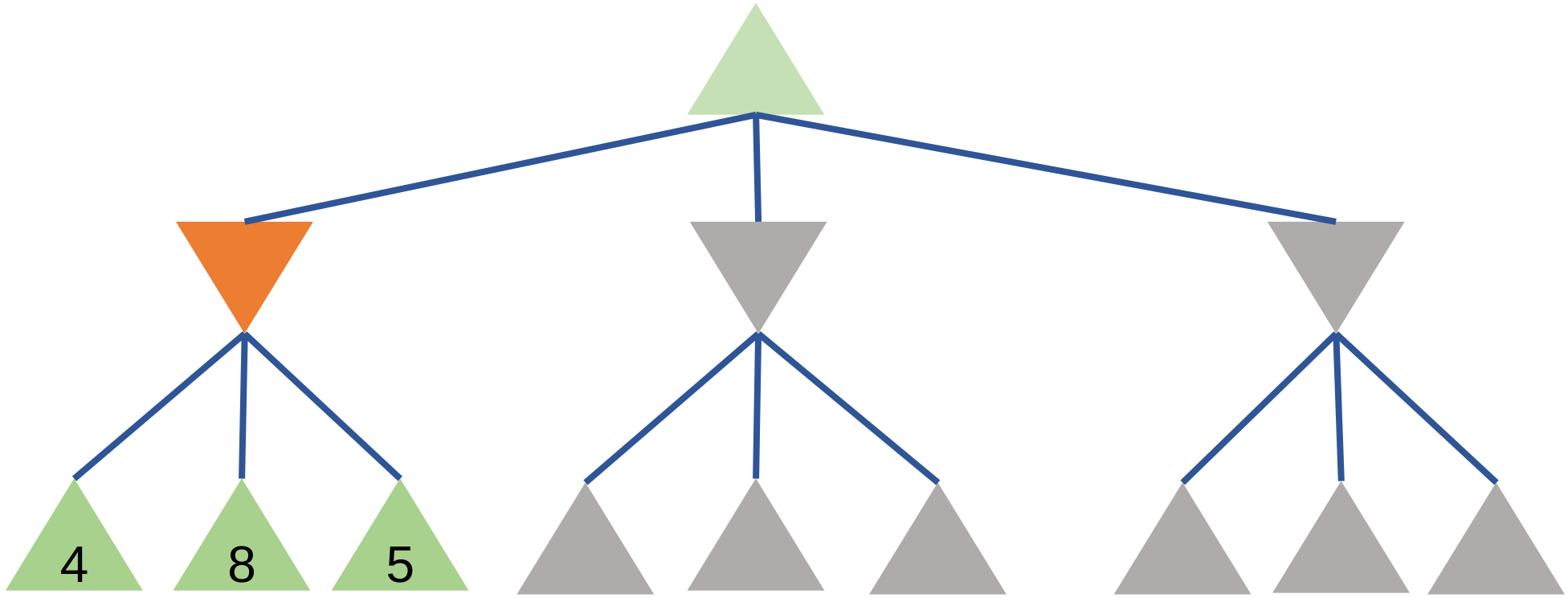
极小极大算法 Minimax

- 两个选手
- 两次行动



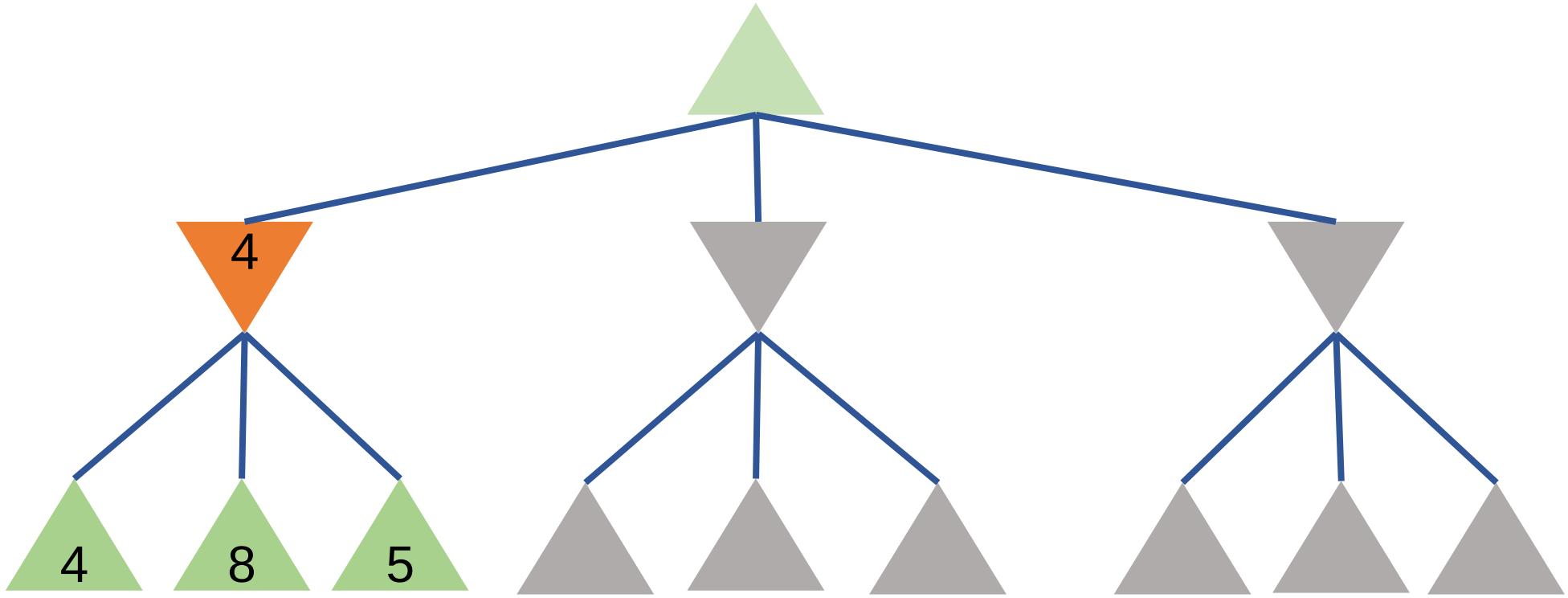
极小极大算法 Minimax

- 两个选手
- 两次行动



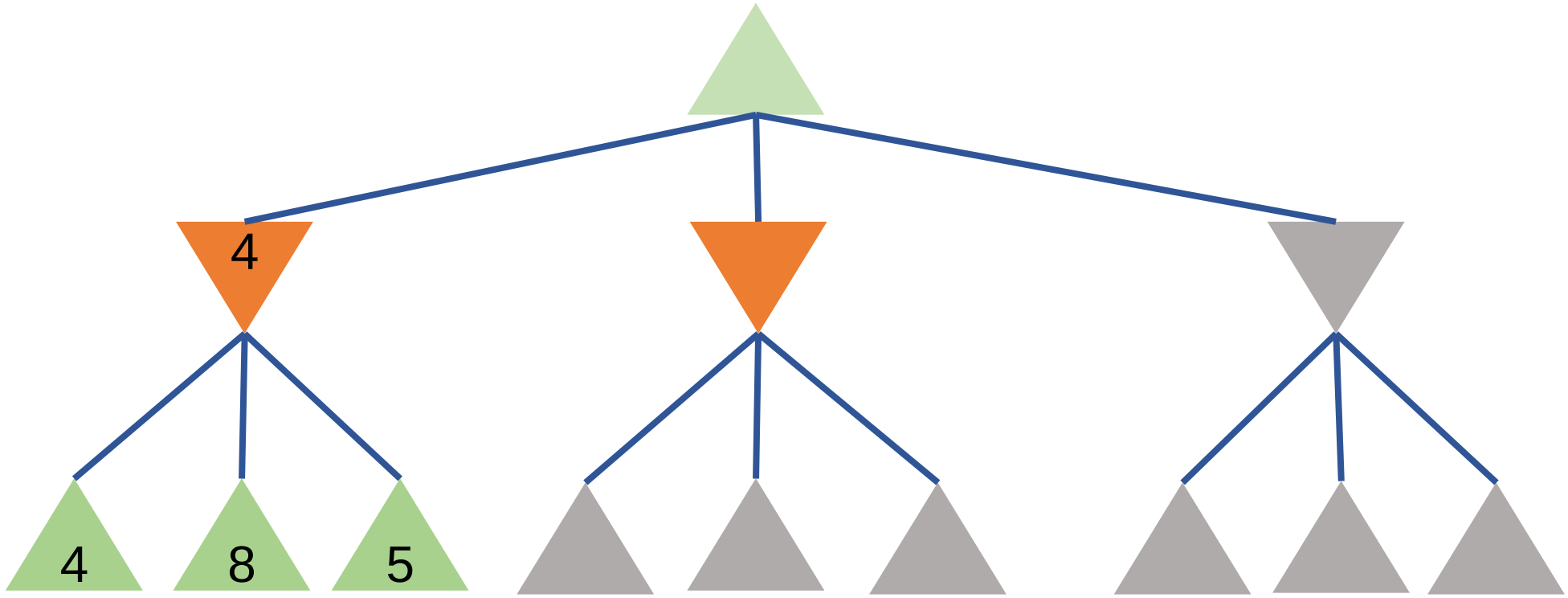
极小极大算法 Minimax

- 两个选手
- 两次行动



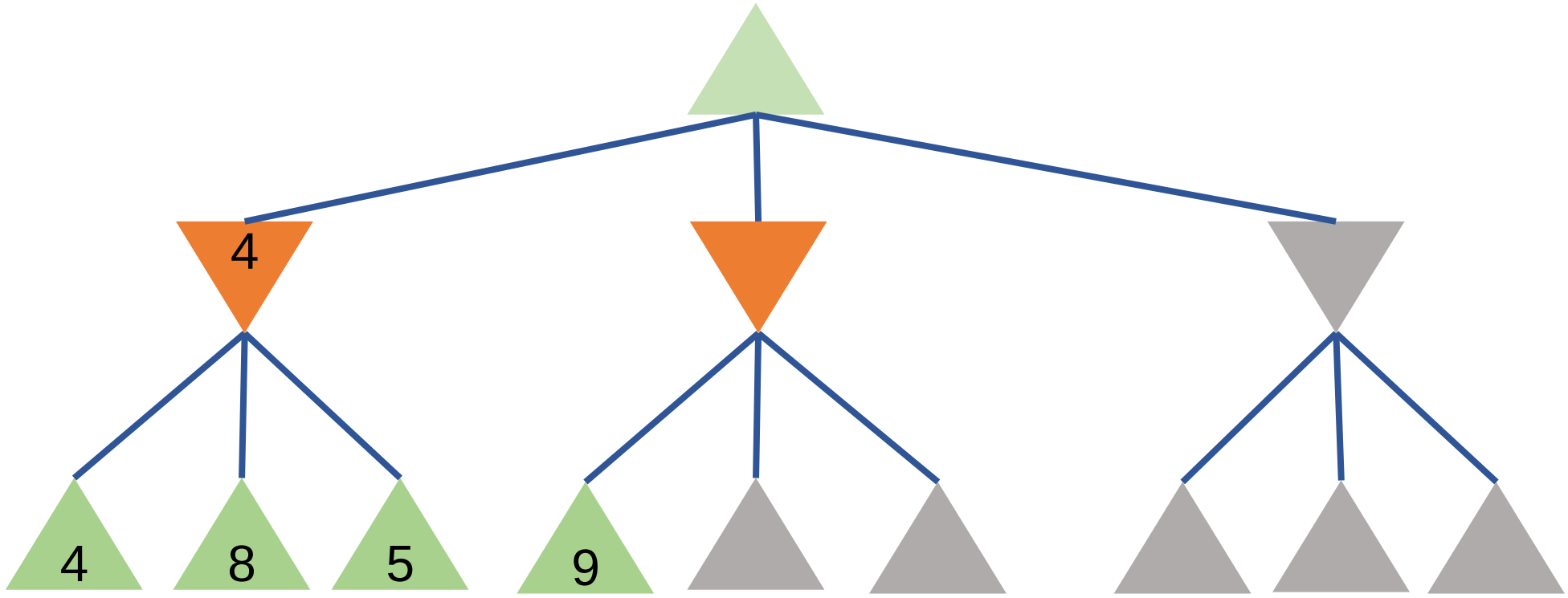
极小极大算法 Minimax

- 两个选手
- 两次行动



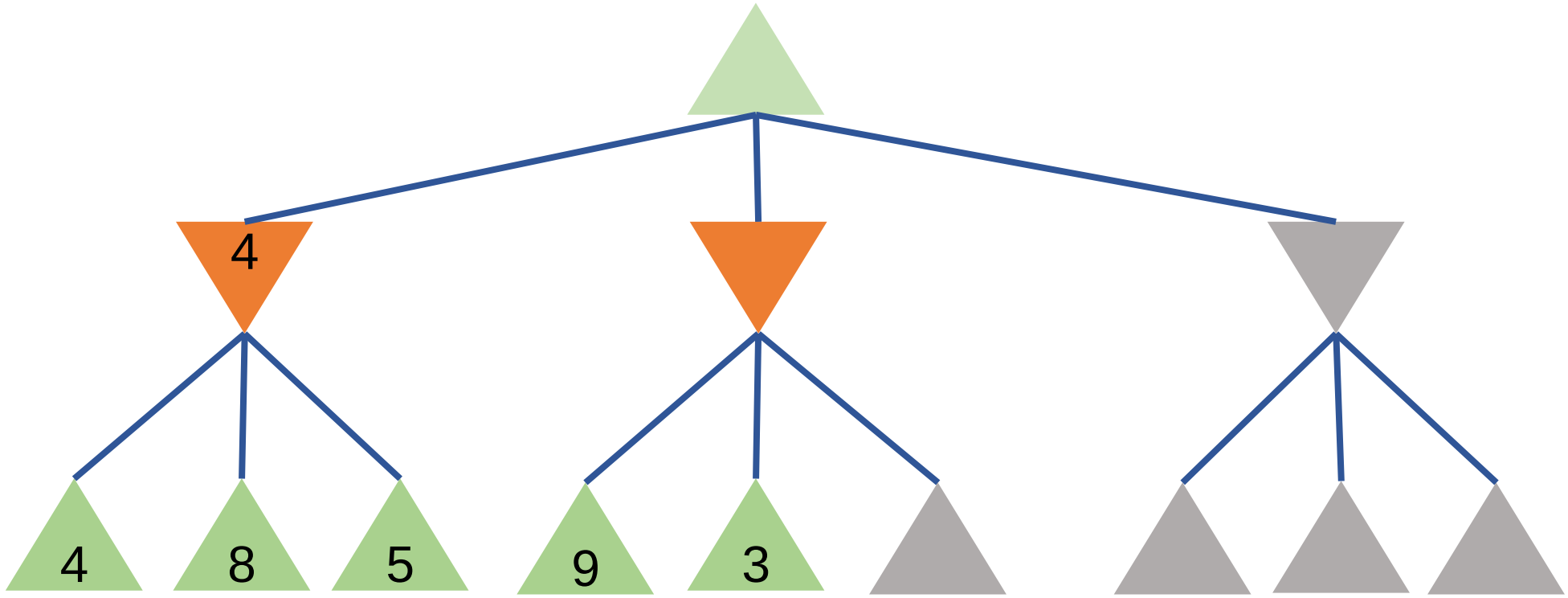
极小极大算法 Minimax

- 两个选手
- 两次行动



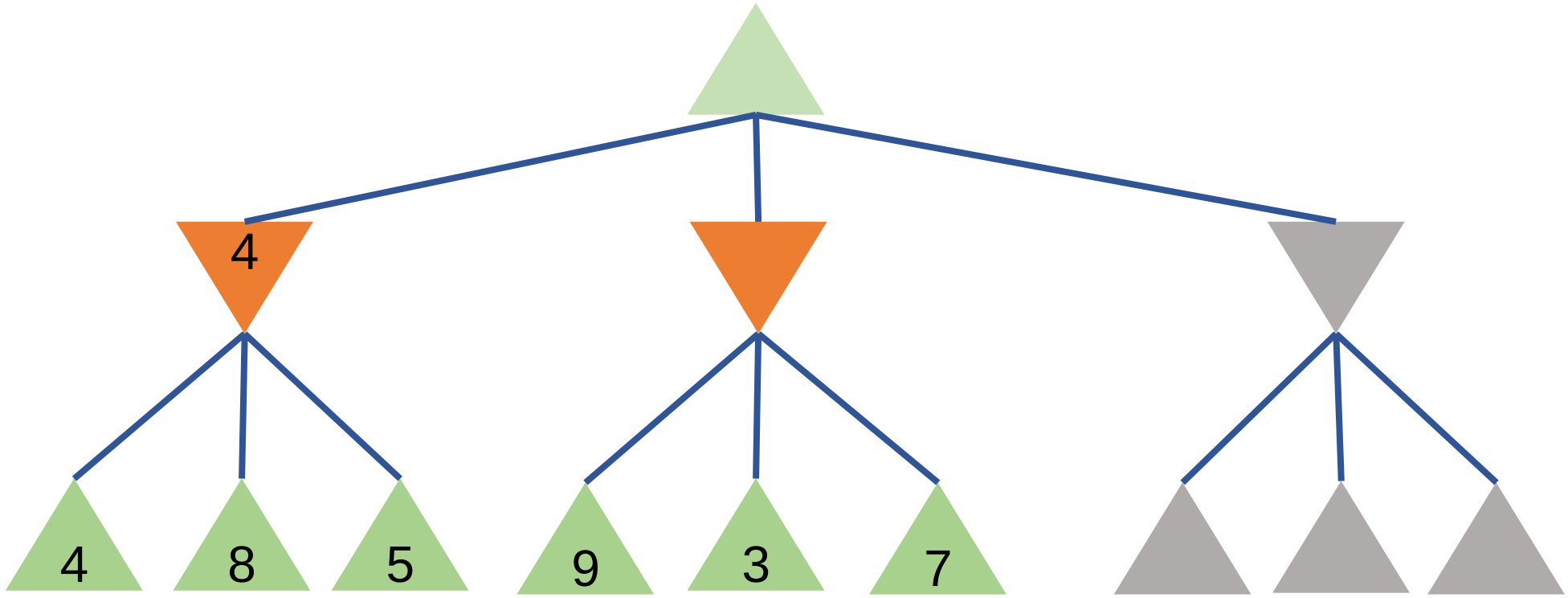
极小极大算法 Minimax

- 两个选手
- 两次行动



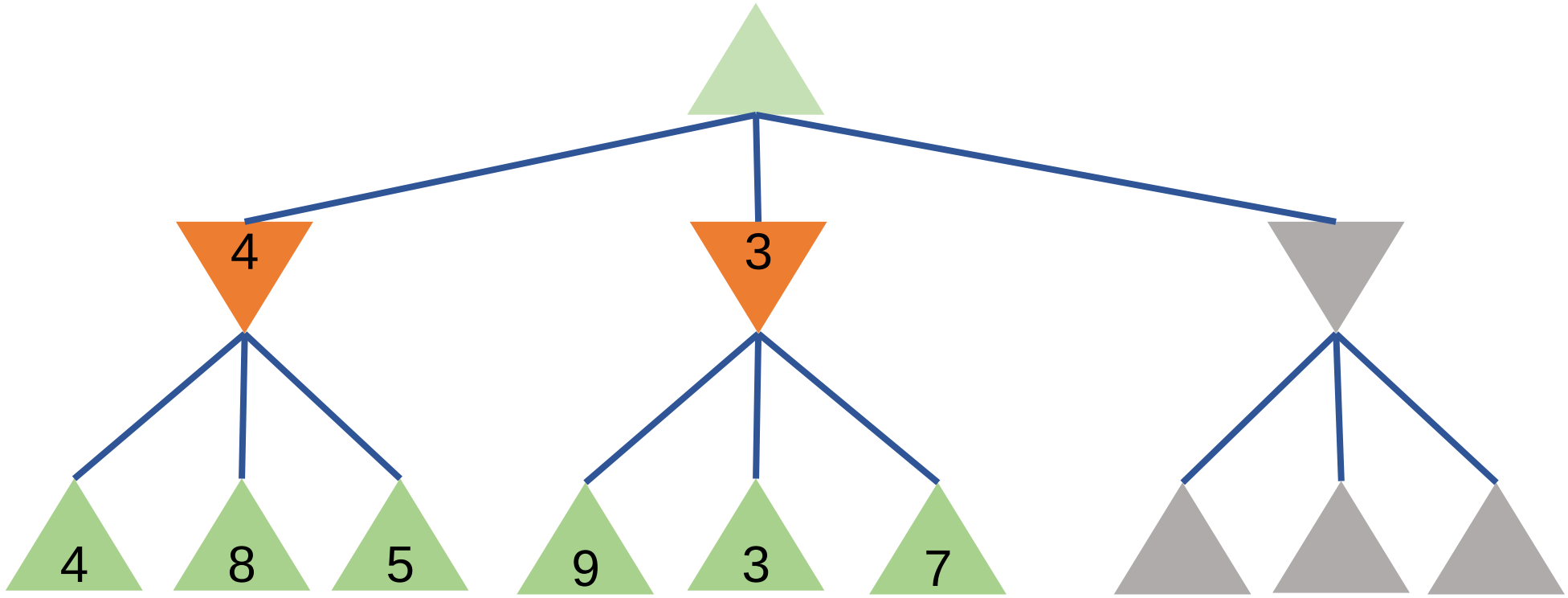
极小极大算法 Minimax

- 两个选手
- 两次行动



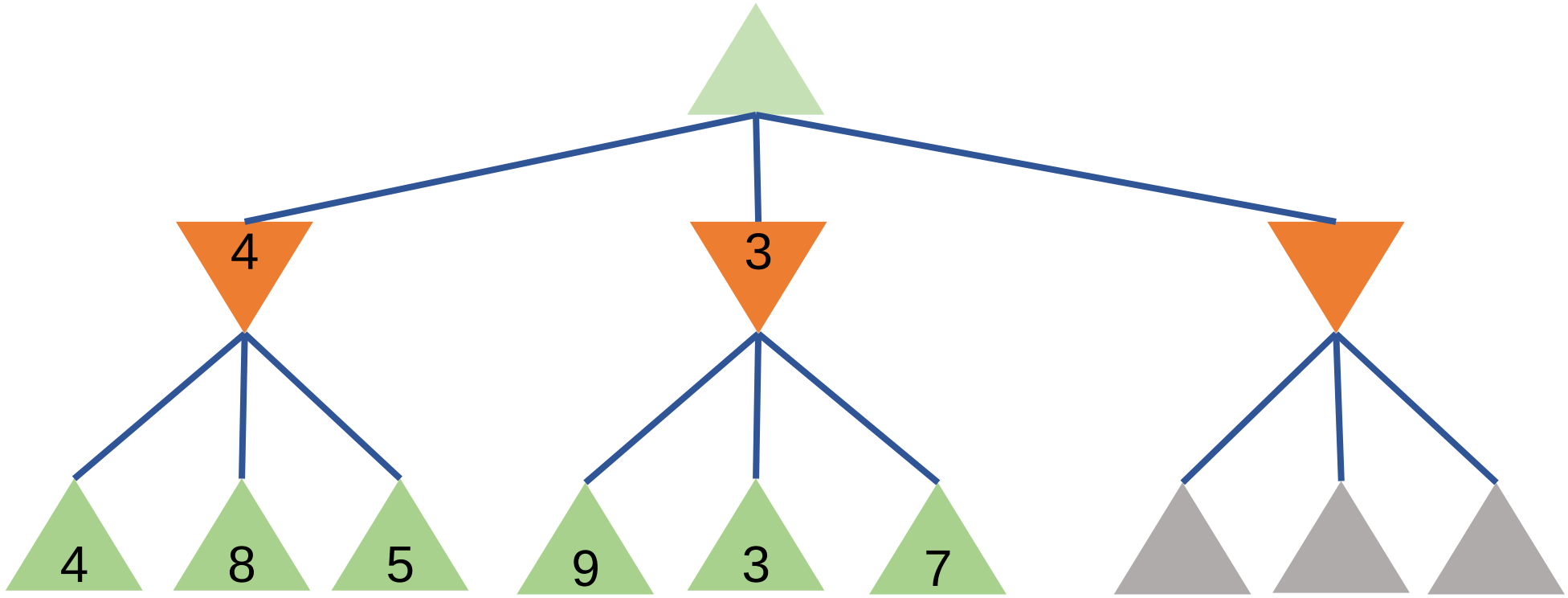
极小极大算法 Minimax

- 两个选手
- 两次行动



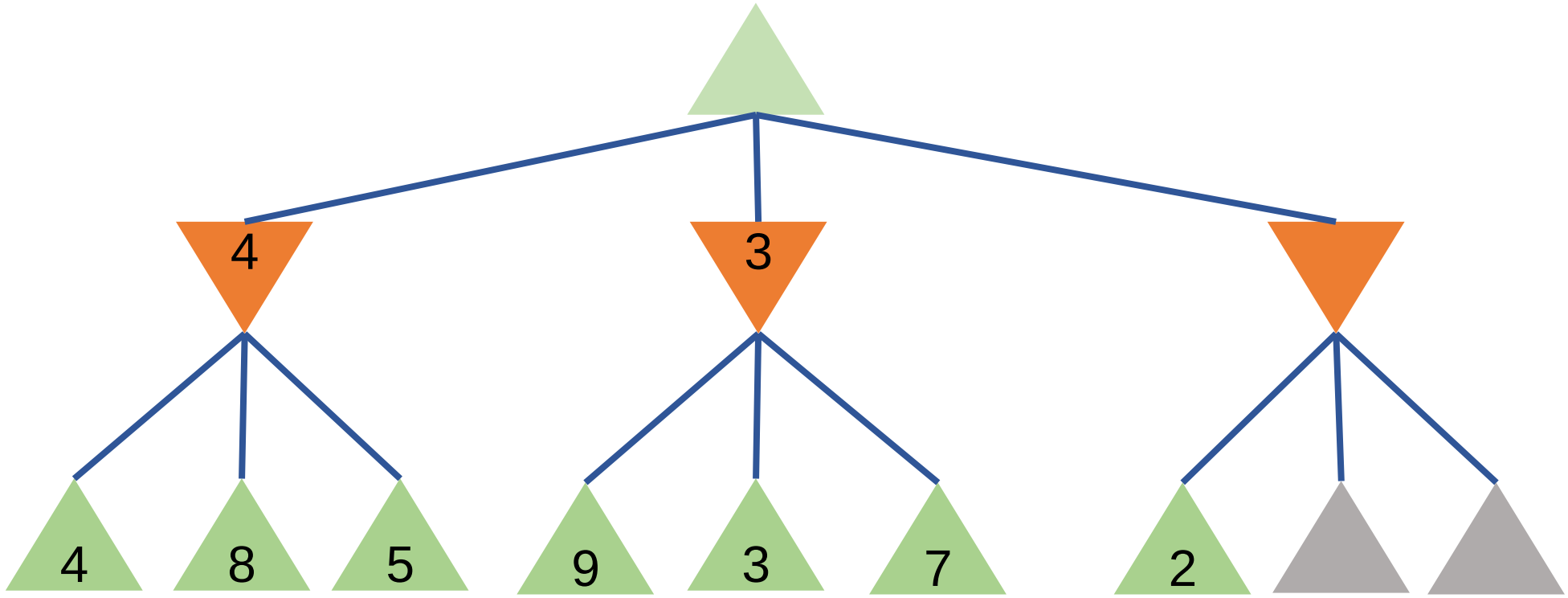
极小极大算法 Minimax

- 两个选手
- 两次行动



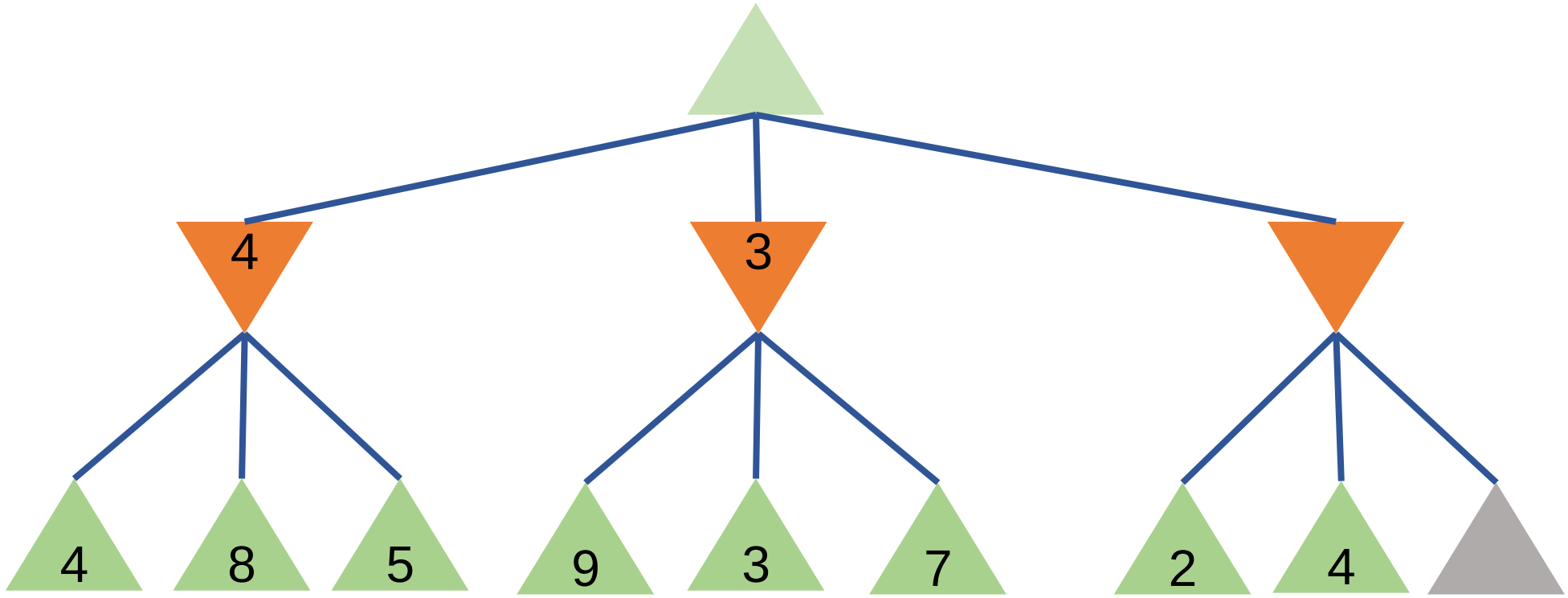
极小极大算法 Minimax

- 两个选手
- 两次行动



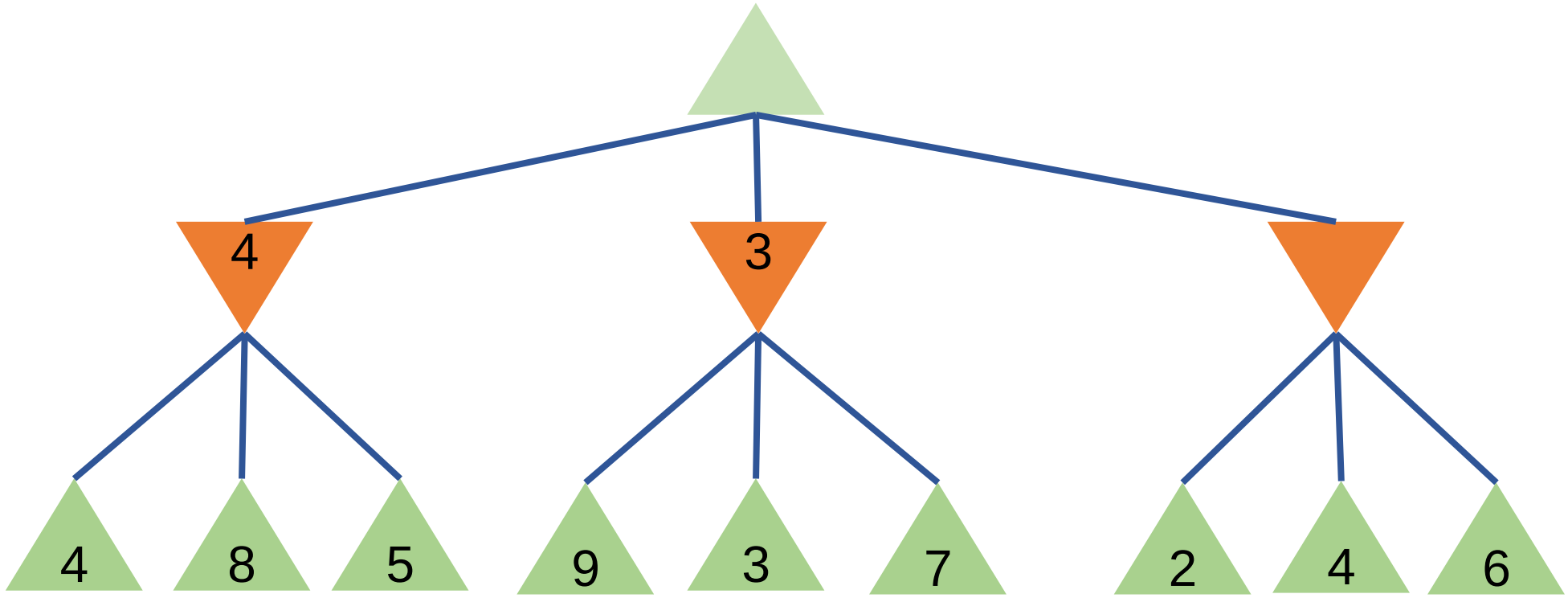
极小极大算法 Minimax

- 两个选手
- 两次行动



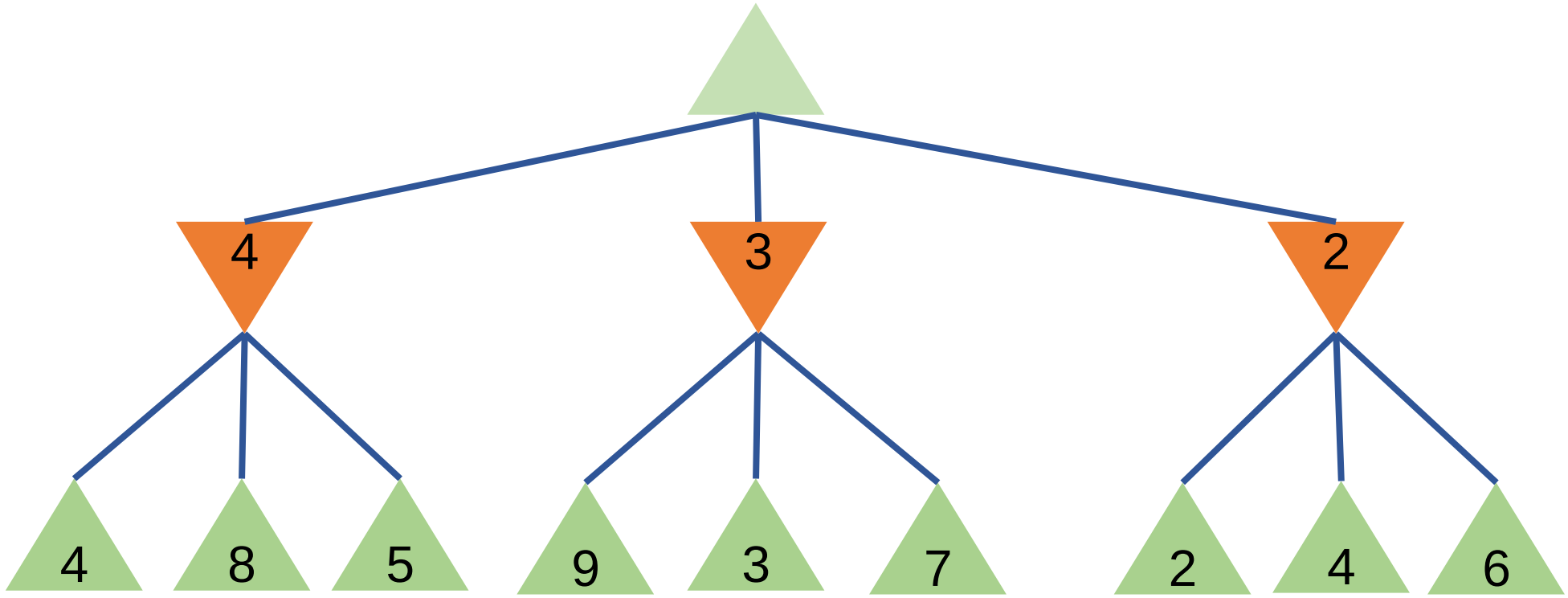
极小极大算法 Minimax

- 两个选手
- 两次行动



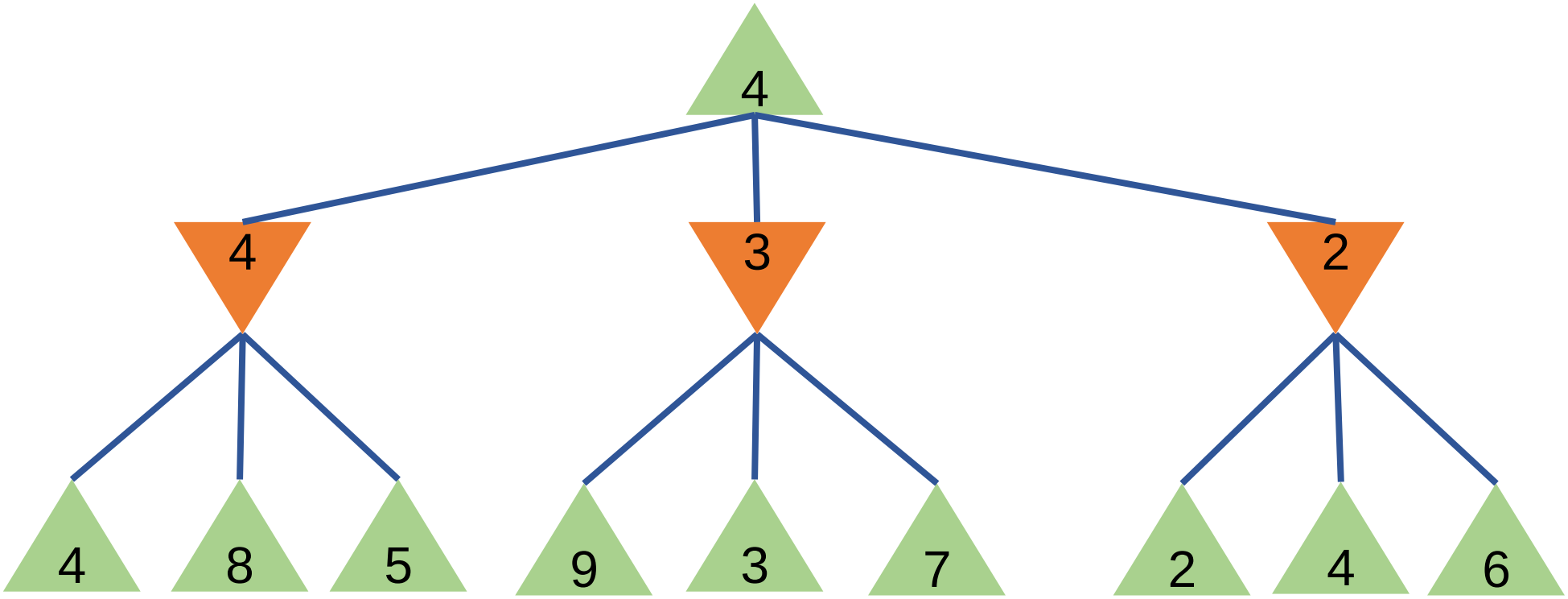
极小极大算法 Minimax

- 两个选手
- 两次行动



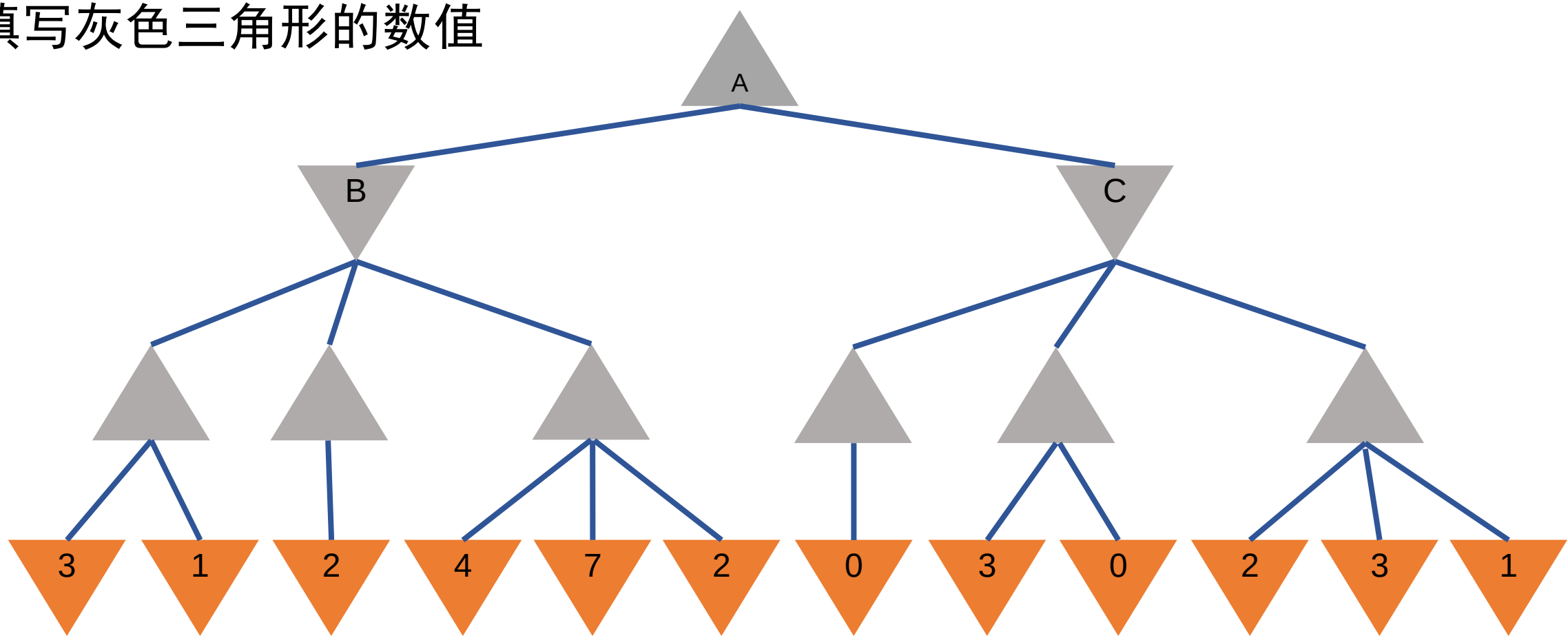
极小极大算法 Minimax

- 两个选手
- 两次行动



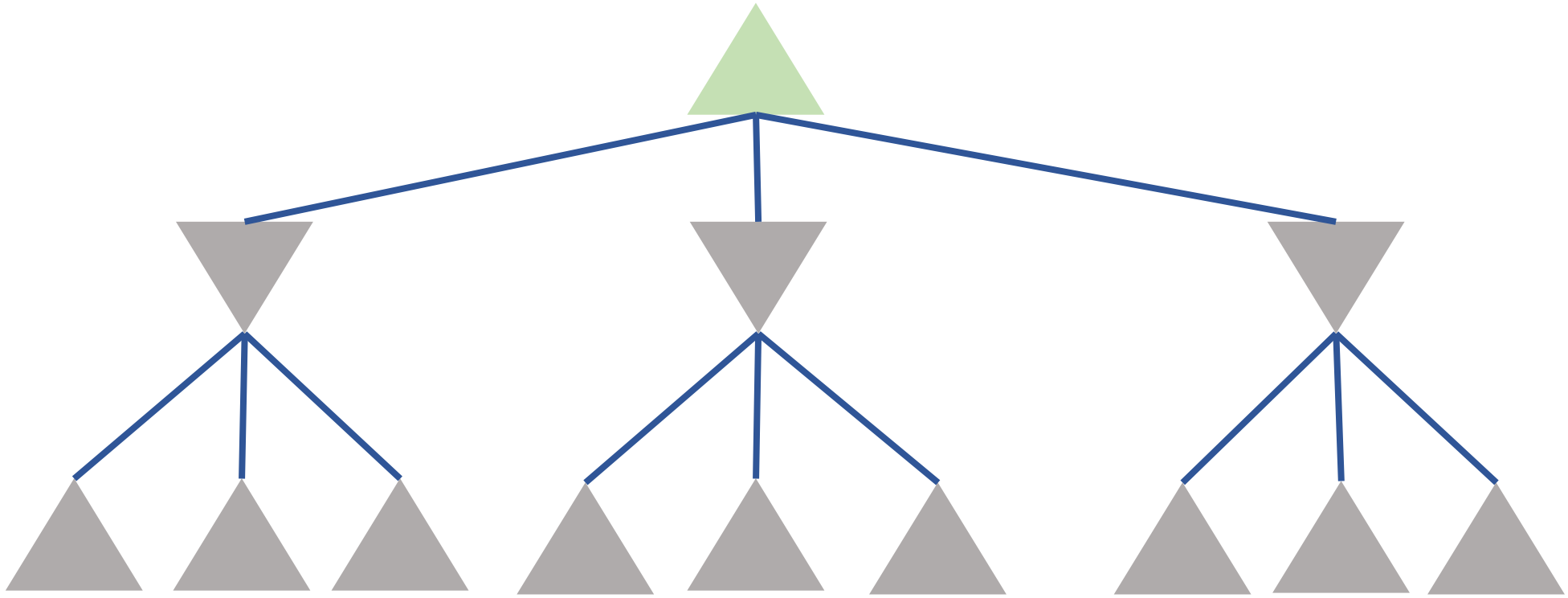
练习 #1

- 填写灰色三角形的数值



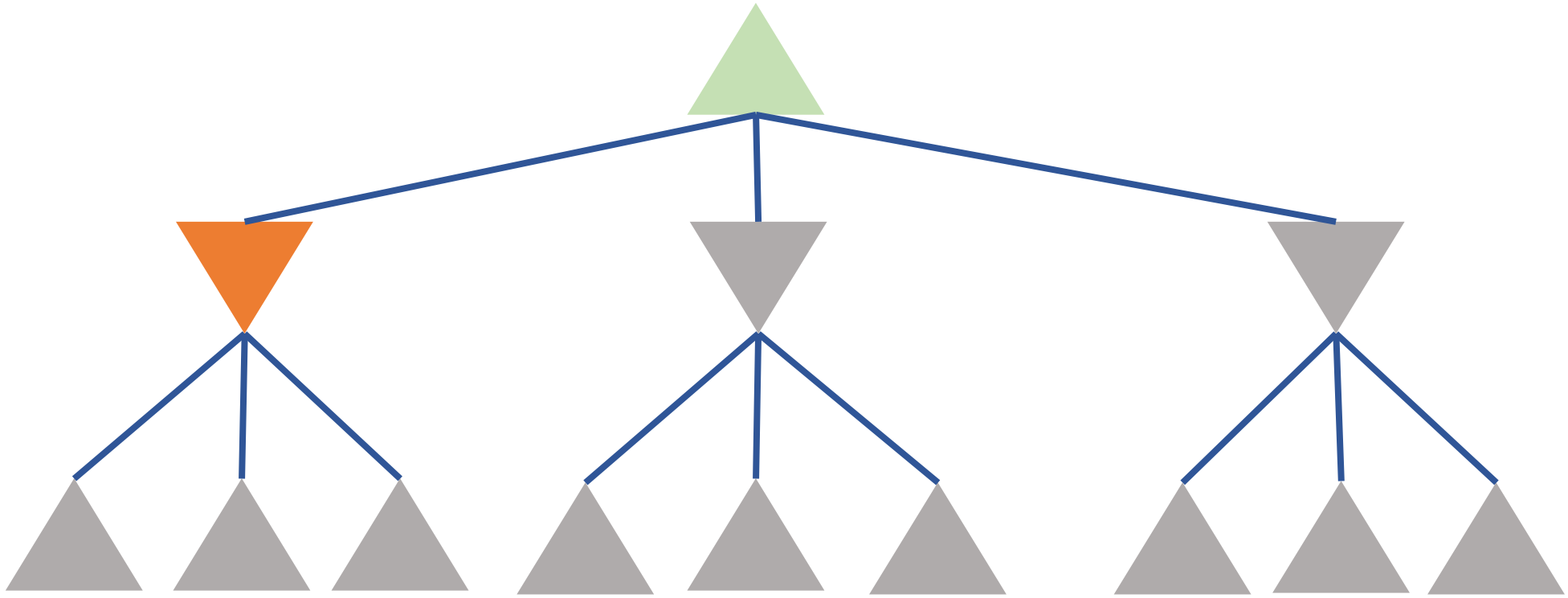
如何改进极小极大算法 Minimax?

- 两个选手
- 两次行动



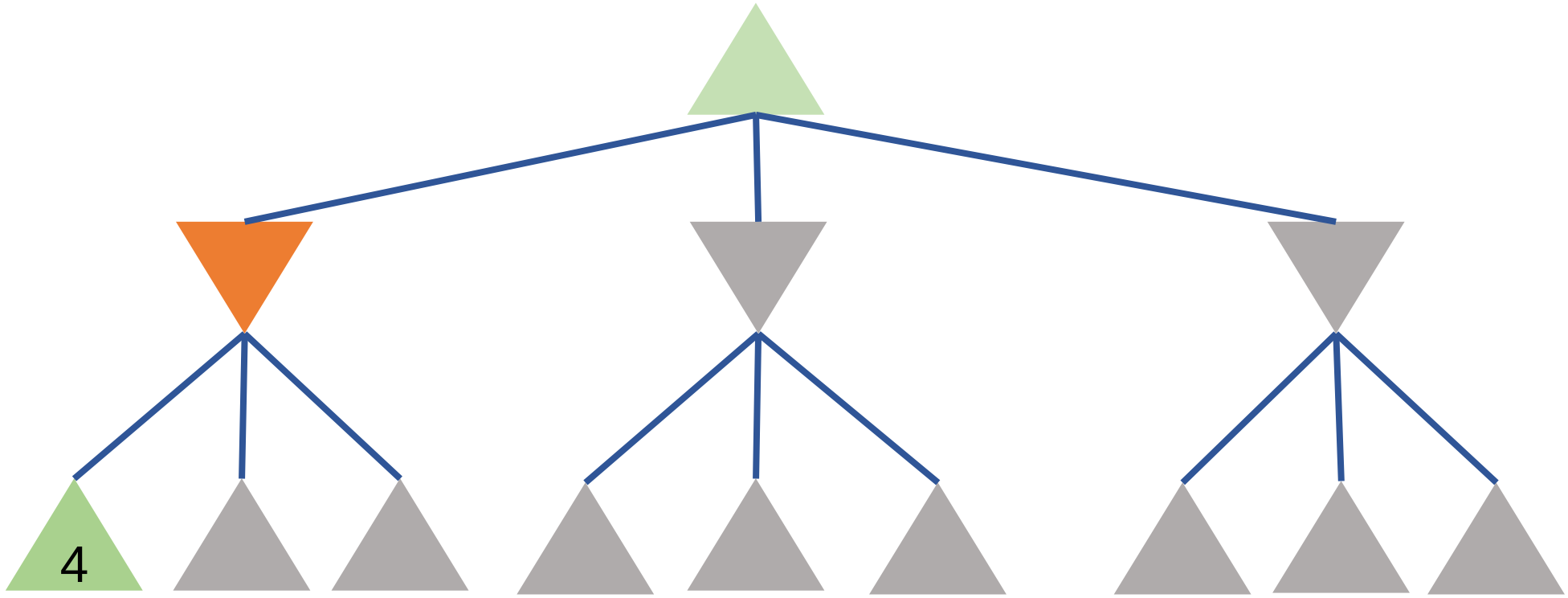
如何改进极小极大算法 Minimax?

- 两个选手
- 两次行动



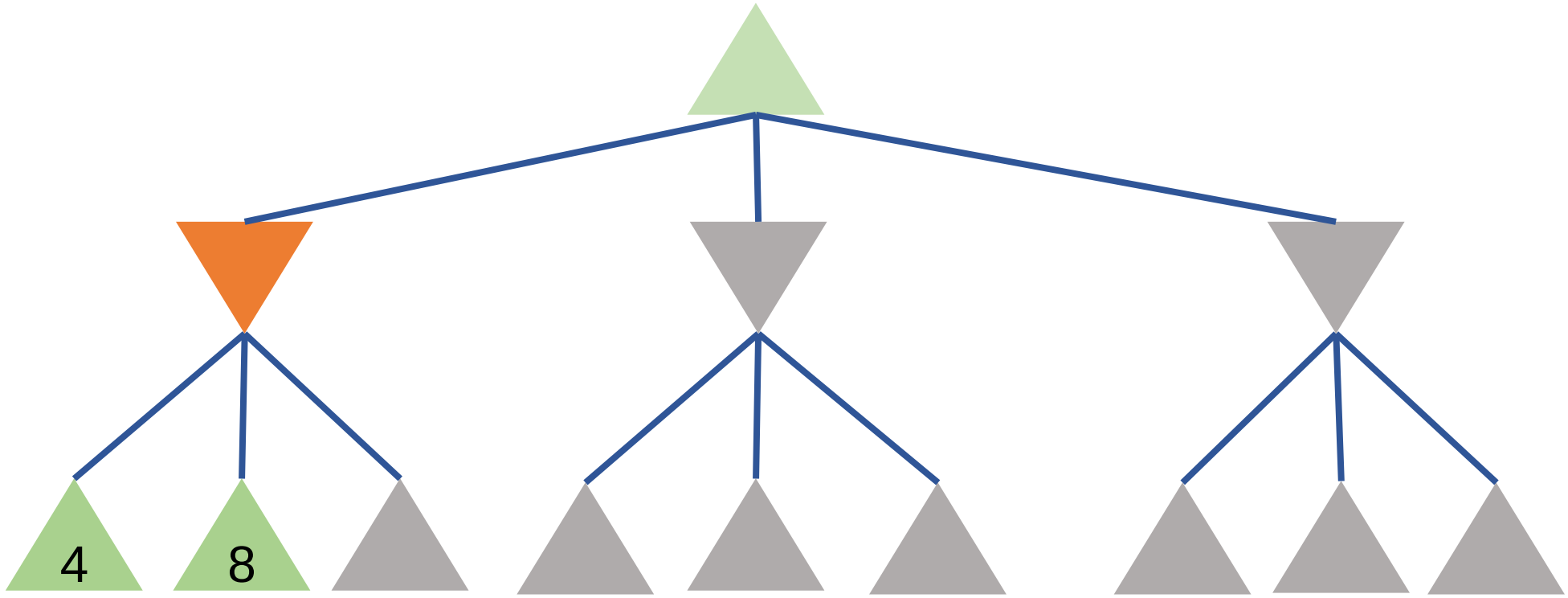
如何改进极小极大算法 Minimax?

- 两个选手
- 两次行动



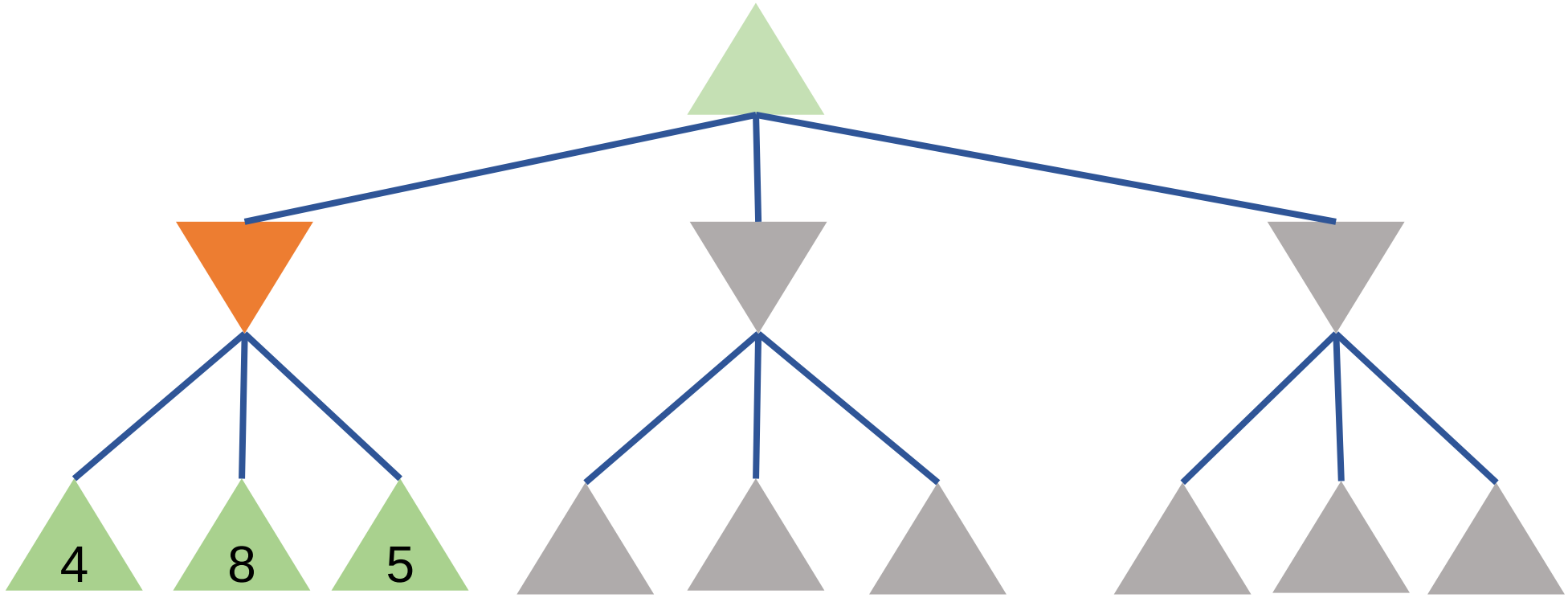
如何改进极小极大算法 Minimax?

- 两个选手
- 两次行动



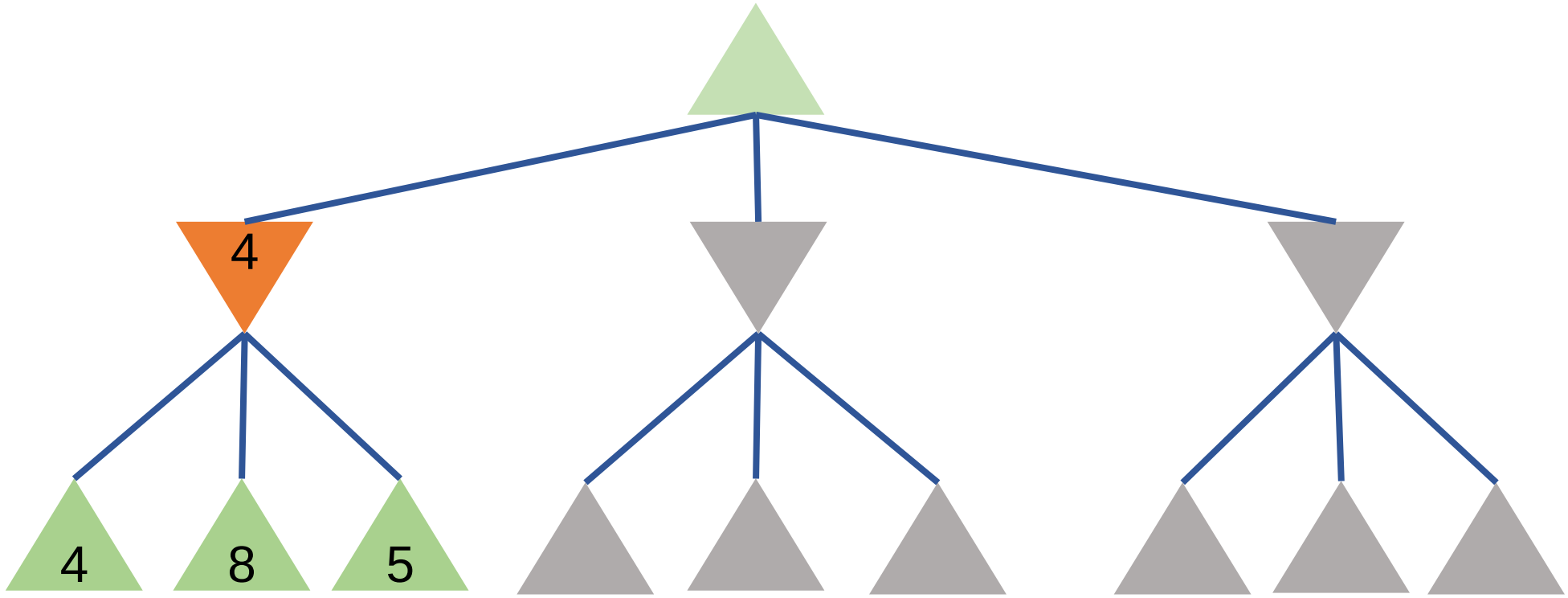
如何改进极小极大算法 Minimax?

- 两个选手
- 两次行动



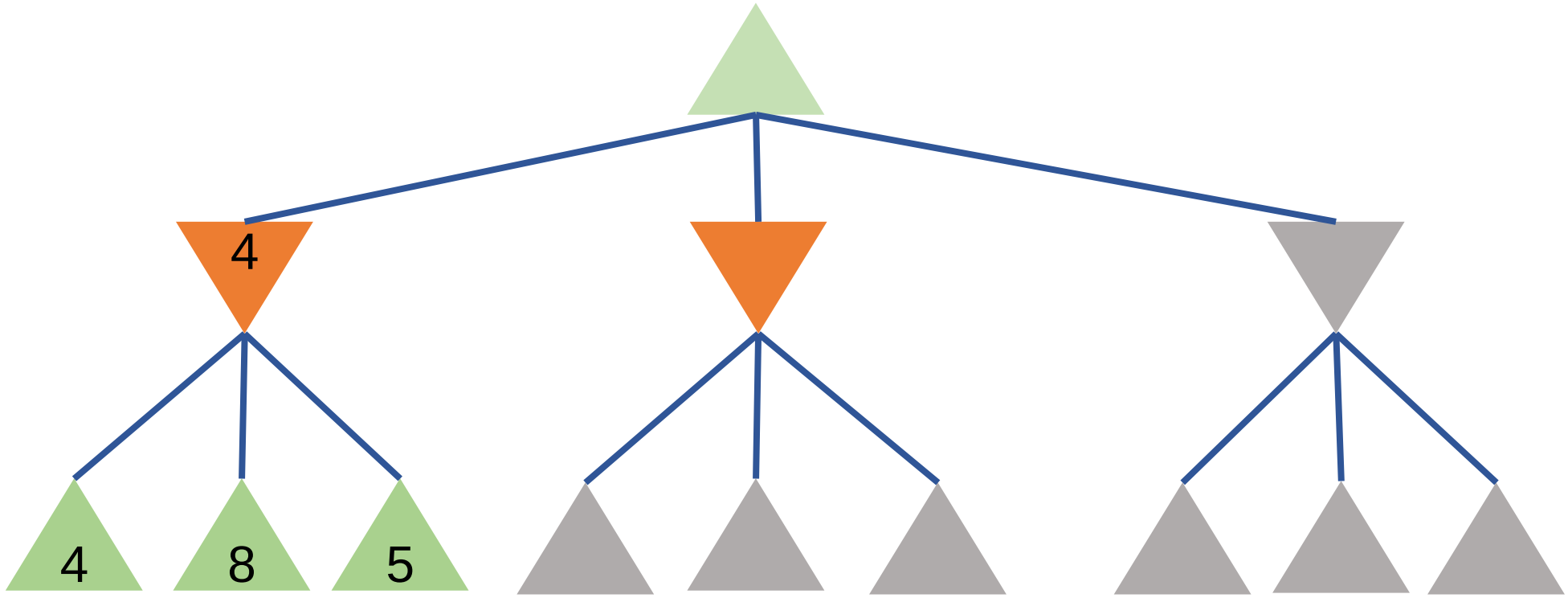
如何改进极小极大算法 Minimax?

- 两个选手
- 两次行动



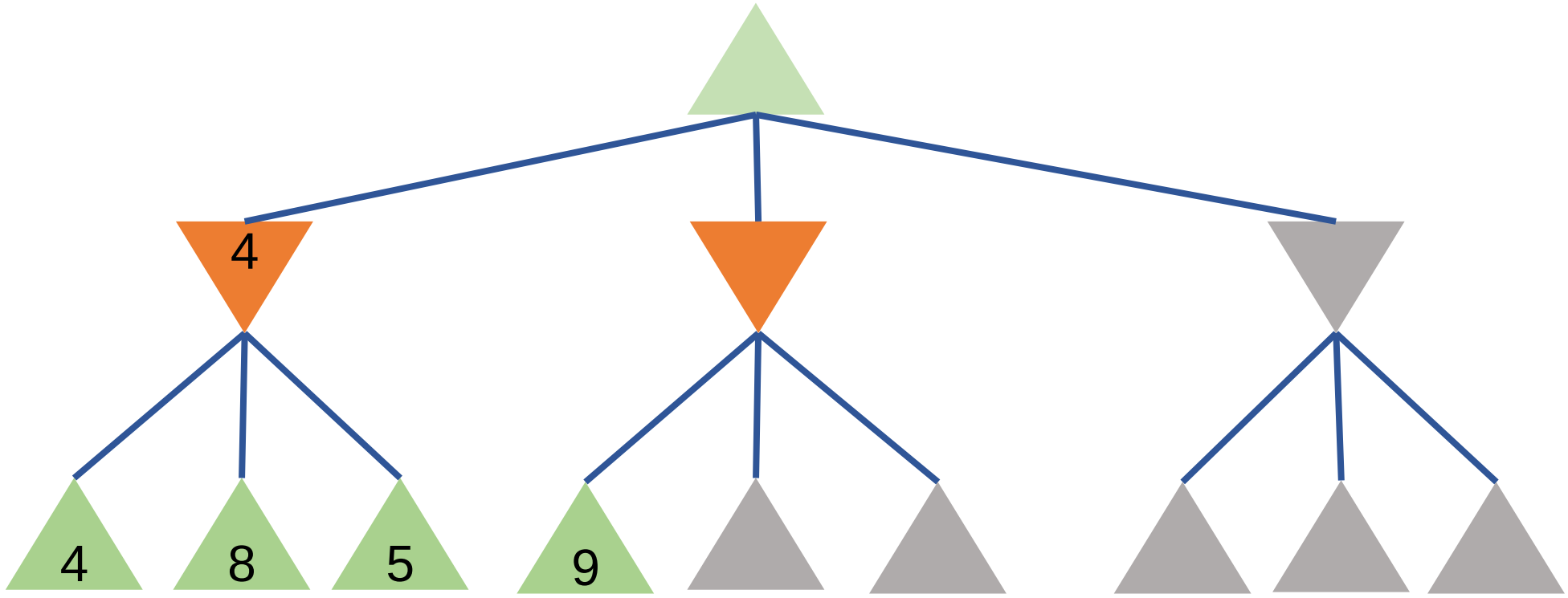
如何改进极小极大算法 Minimax?

- 两个选手
- 两次行动



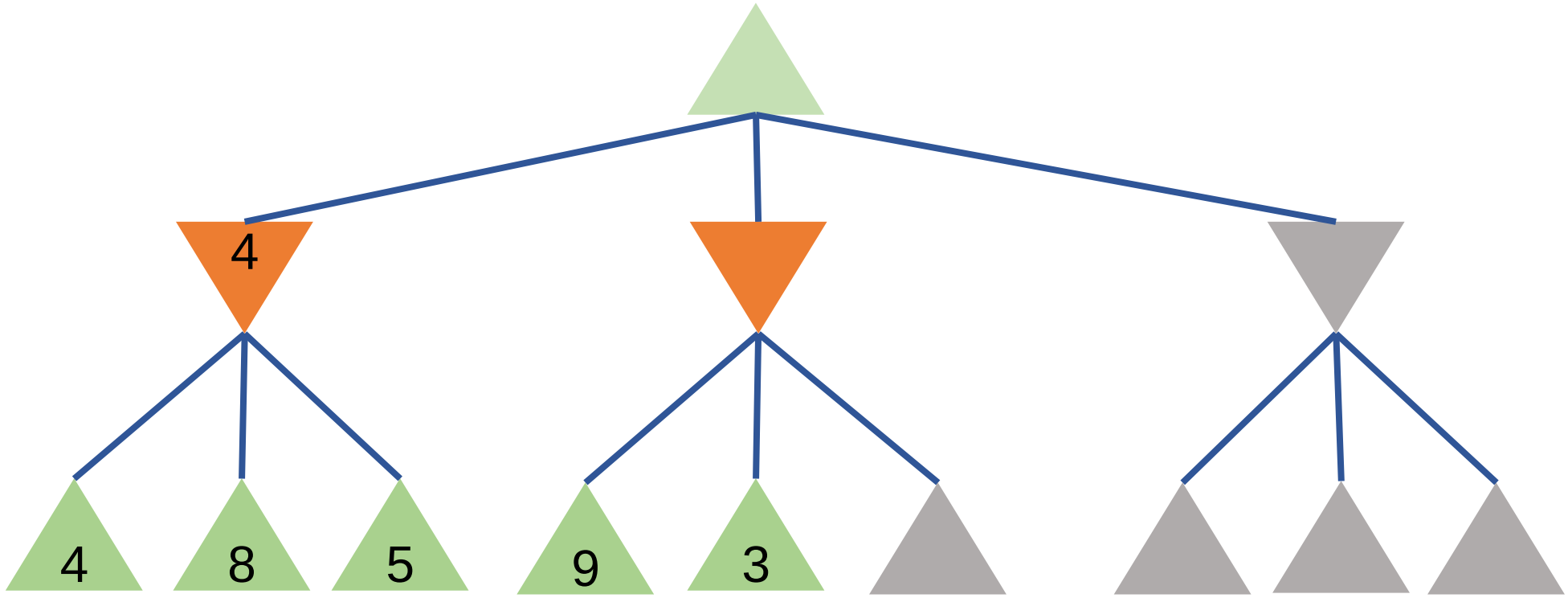
如何改进极小极大算法 Minimax?

- 两个选手
- 两次行动



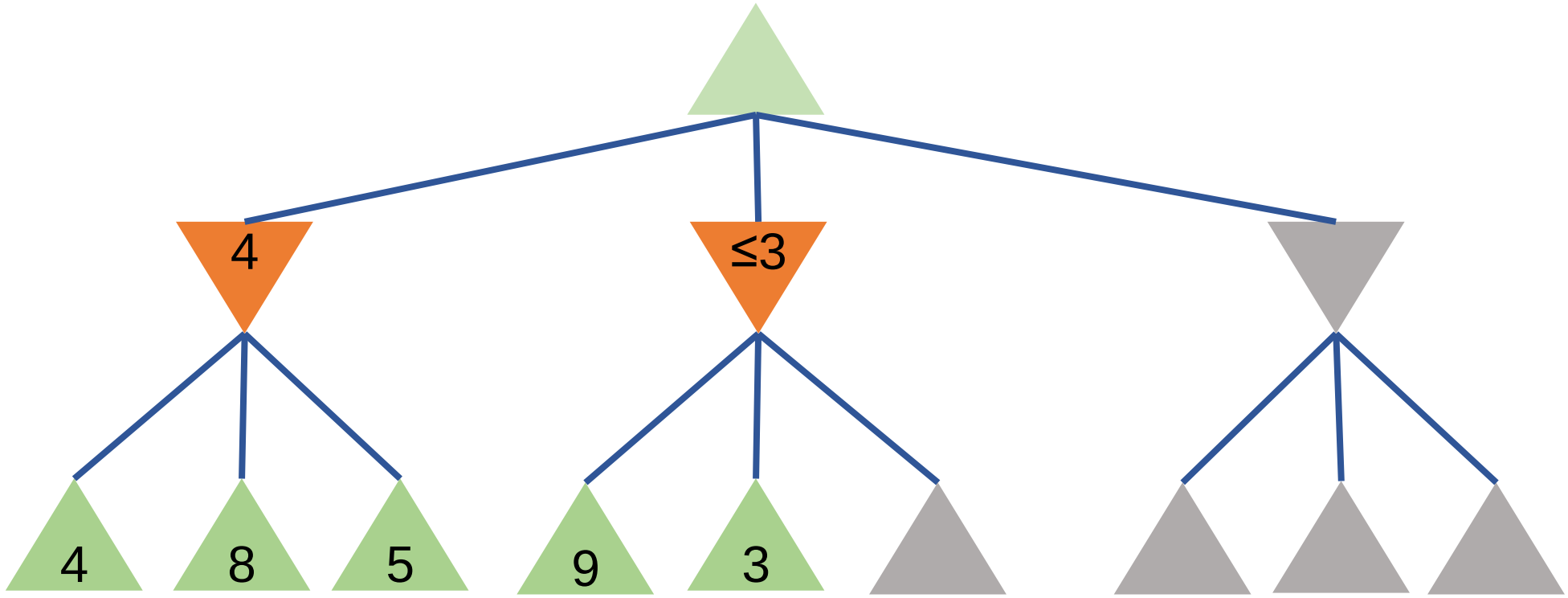
如何改进极小极大算法 Minimax?

- 两个选手
- 两次行动



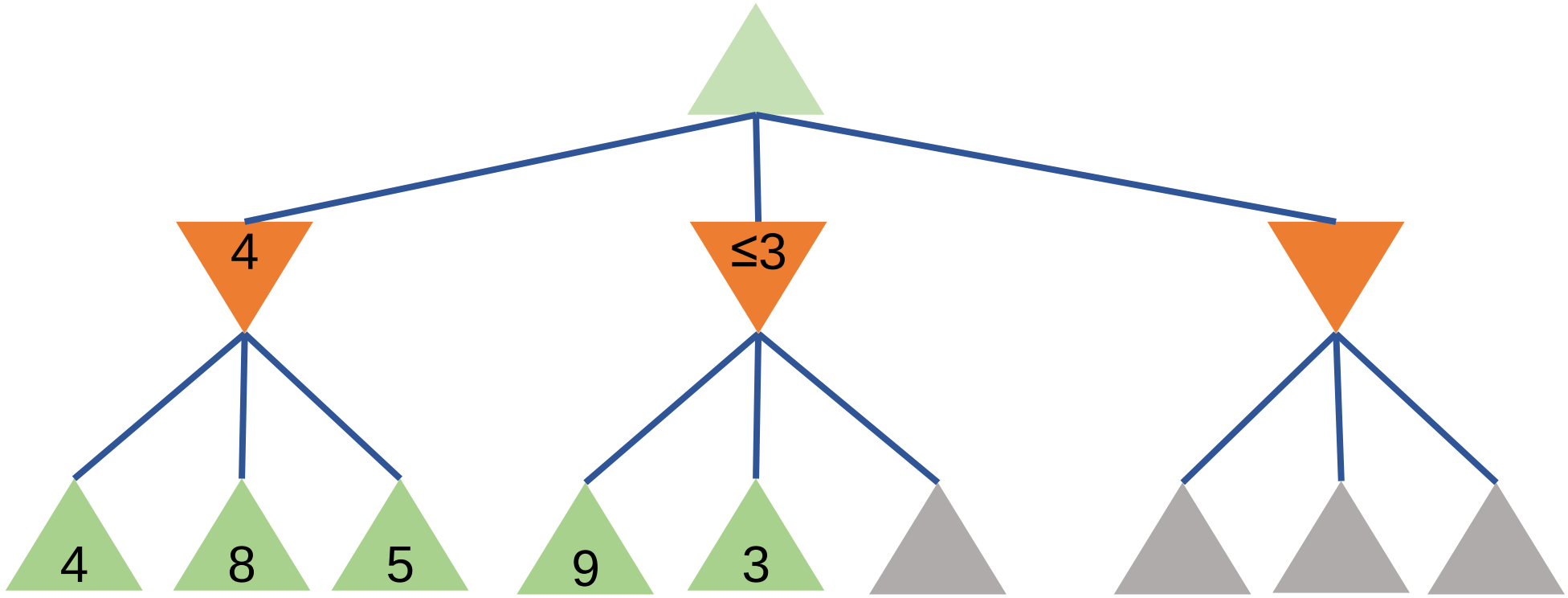
如何改进极小极大算法 Minimax?

- 两个选手
- 两次行动



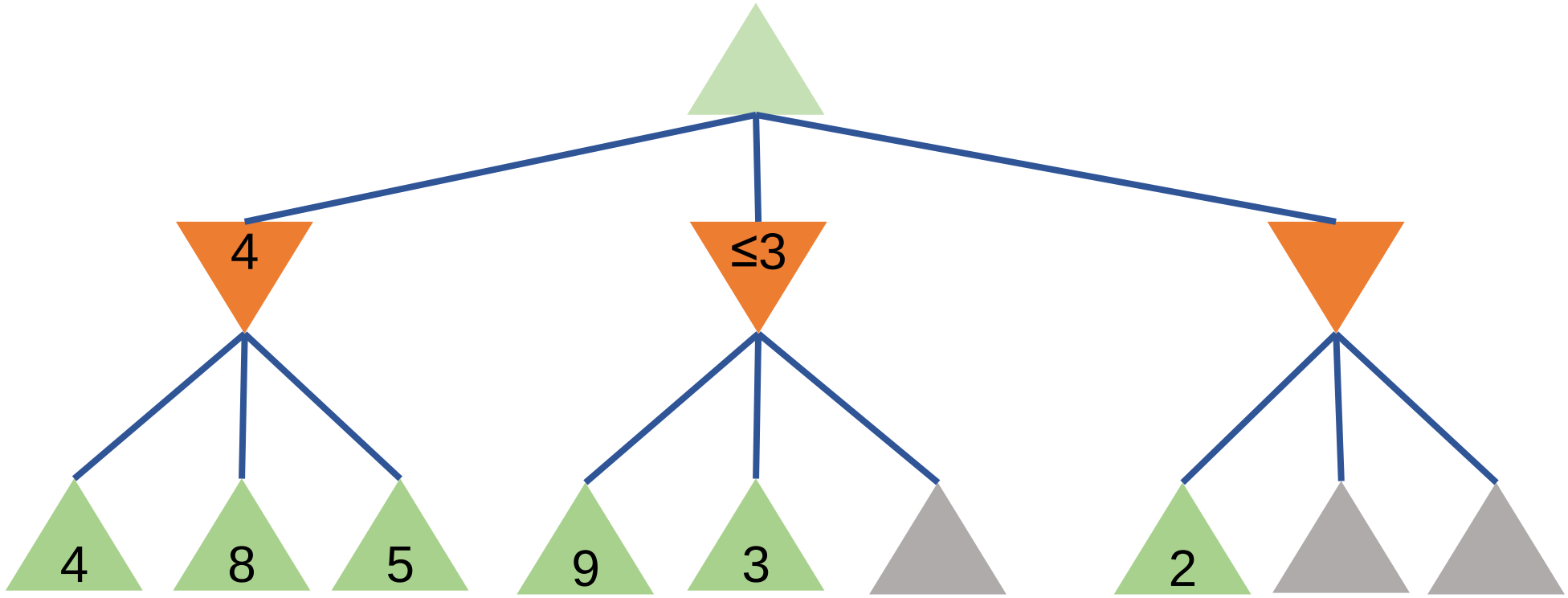
如何改进极小极大算法 Minimax?

- 两个选手
- 两次行动



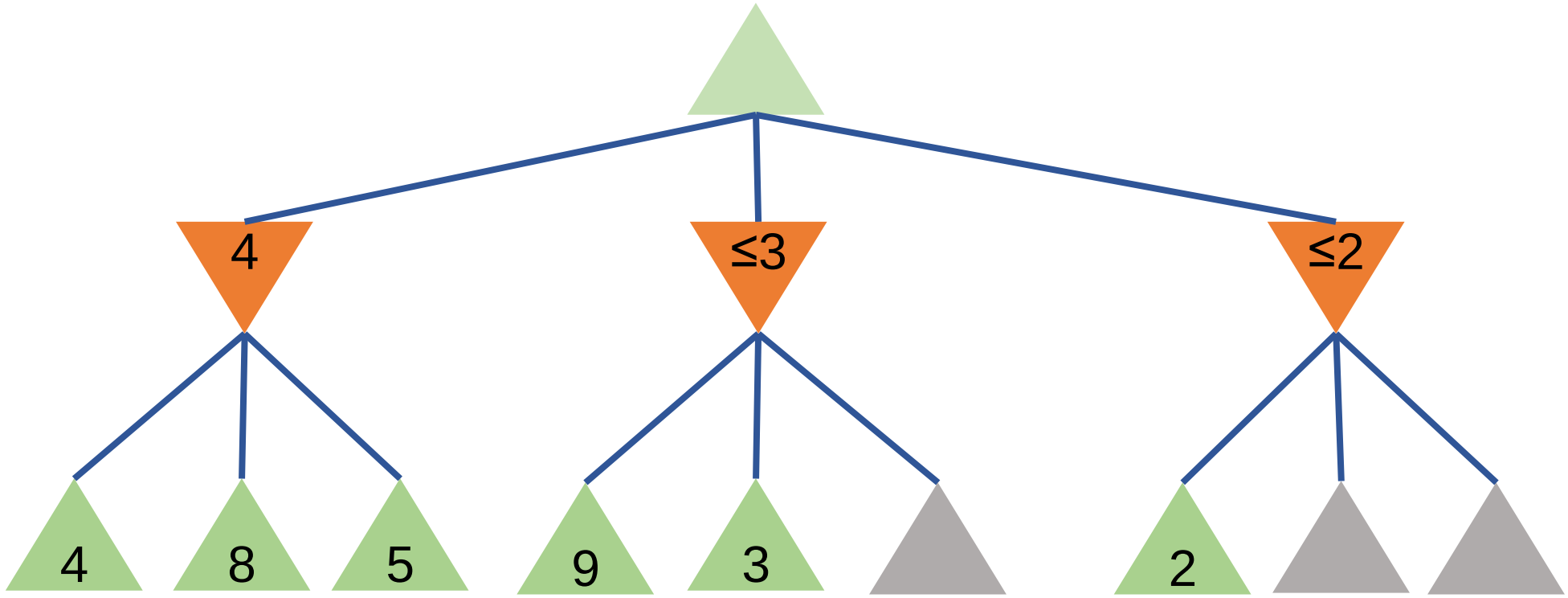
如何改进极小极大算法 Minimax?

- 两个选手
- 两次行动



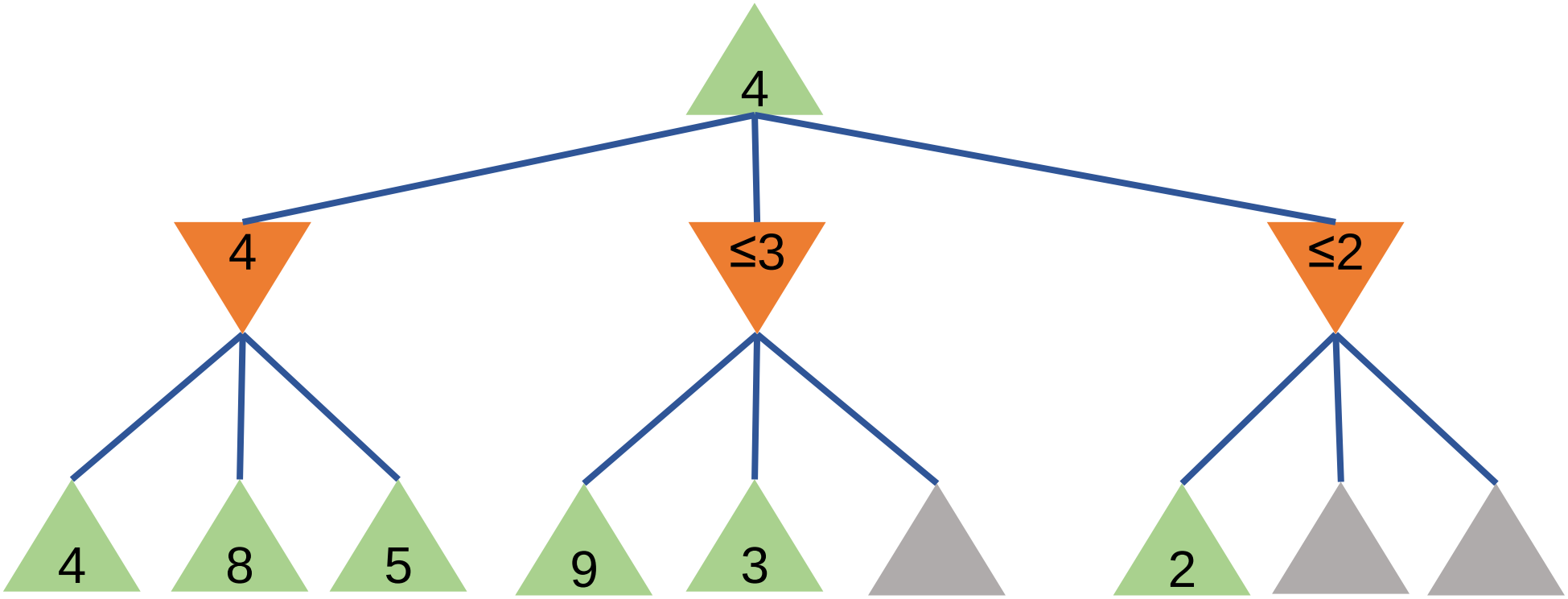
如何改进极小极大算法 Minimax?

- 两个选手
- 两次行动



如何改进极小极大算法 Minimax?

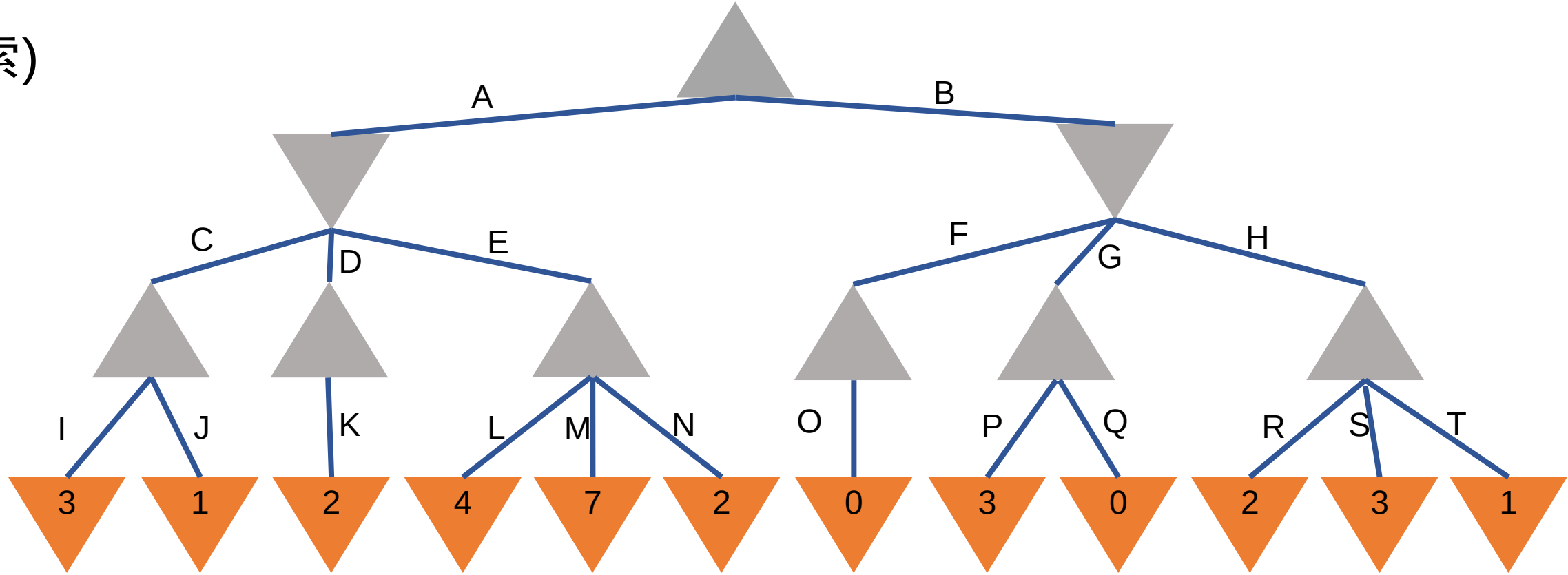
- 两个选手
- 两次行动



- **Alpha-Beta剪枝 Alpha-Beta Pruning**

练习 #2

- 填写灰色三角形的数值并注明哪些分支会被剪枝剪掉(从左至右探索)



行动次数

X	O	X
O	O	
X	X	X

255,168

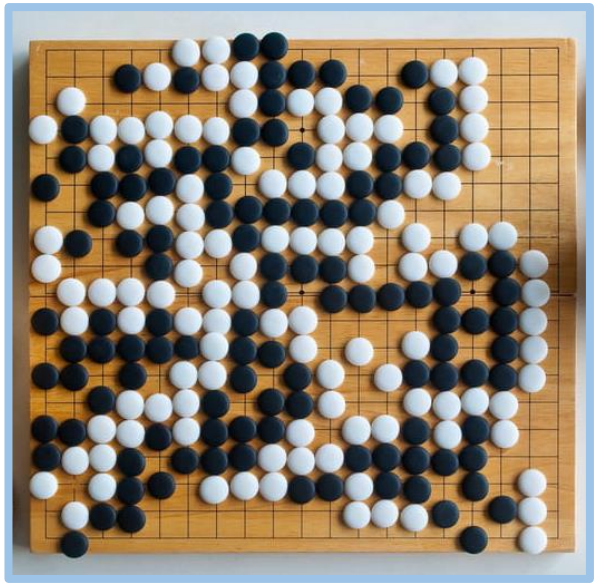
10^{29000}

深度限制极小极大算法 Depth-limited Minimax

- Minimax 是深度不受限的:从初始状态到结束状态
- 深度限制极小极大算法 Depth-limited minimax:
  - 改编自Minimax 算法，但是在仅考虑**提前设定好**的行动次数（例，10 次行动），而不会到达结束状态
  - 评估函数 Evaluation function
    - 估计某个状态的**预期效用(expected utility)**
    - -1 : 白子获胜
    - 1: 黑子获胜
    - 0.8: 黑子大概率获胜
    - **决定极大极小算法的表现的好坏**

深度限制算法的计算量

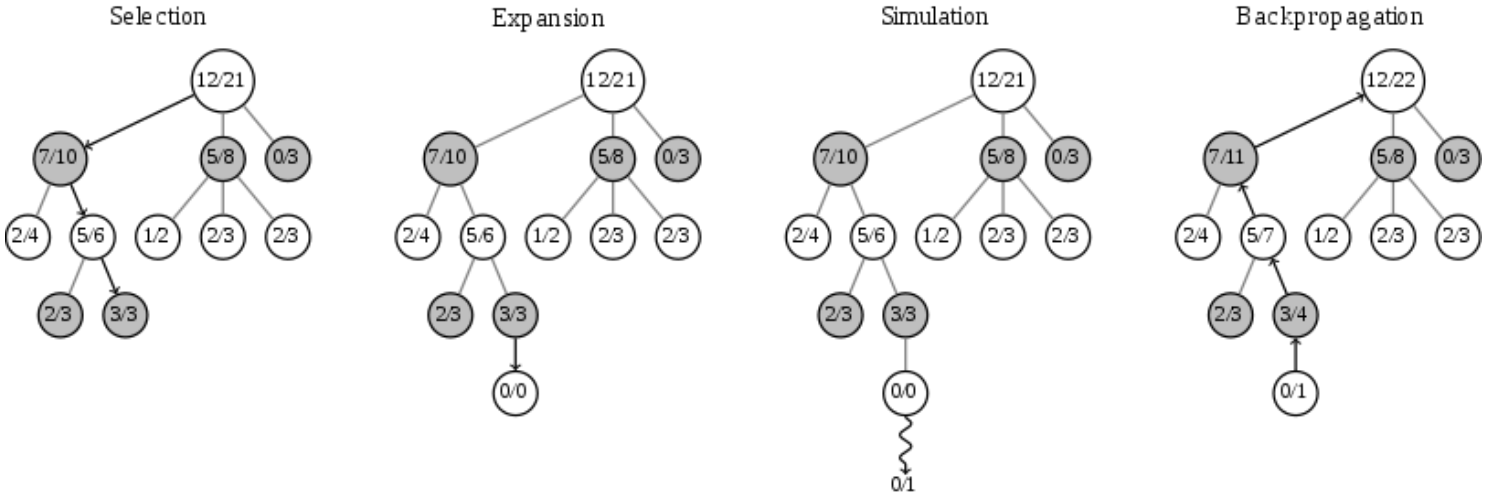
如果深度限制为8
每一步需要考虑1,000,000次

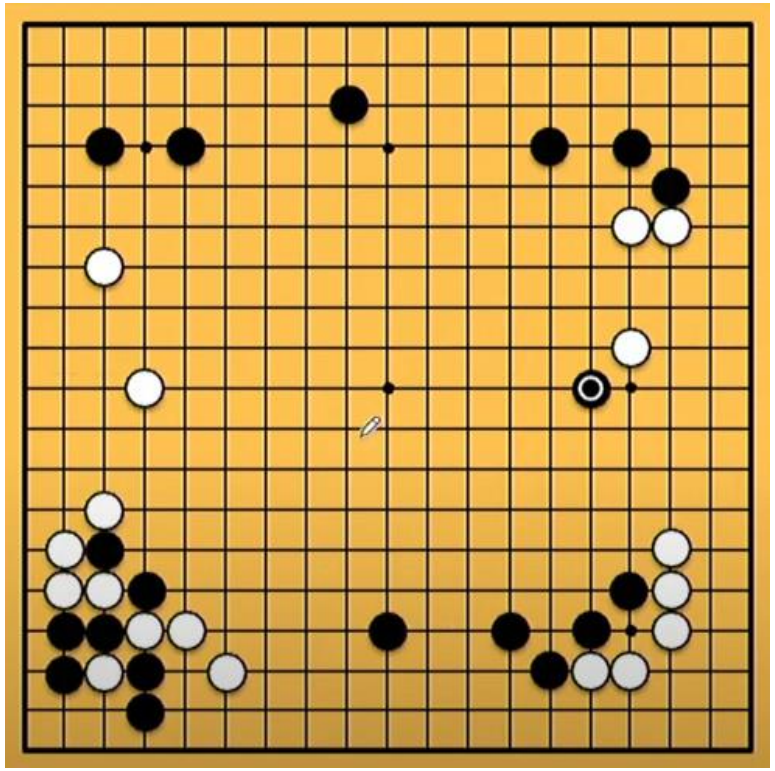


如果深度限制为8
每一步需要考虑8,000,000,000次

蒙特卡洛树搜索 Monte Carlo Tree Search

- 评估函数 Evaluation function → 蒙特卡洛方法
  - playout/rollout：双方在某个状态下**随机(或基于特定选择)**行动，走到结束状态为止，随机很多次(比如一万盘)
  - 计算胜率，胜率代表这个状态的预期效用。





Move 37: AlphaGo vs Lee Sedol

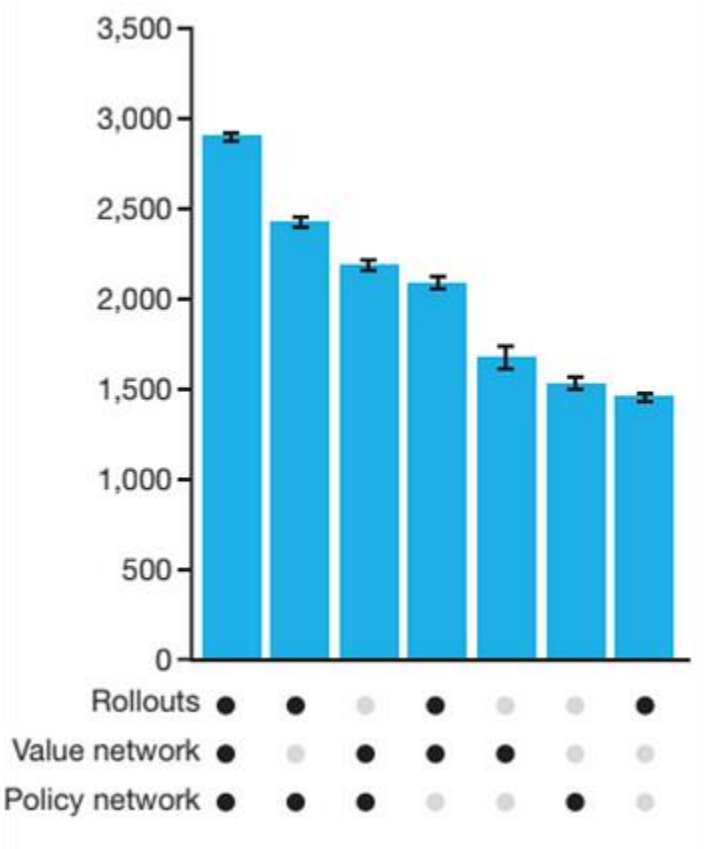
AlphaGo

- MCTS：Rollout
  - ELO rating：衡量零和游戏其中一方的强度

Extended Data Table 7 | Results of a tournament between different variants of AlphaGo

Short name	Policy network	Value network	Rollouts	Mixing constant	Policy GPUs	Value GPUs	Elo rating
α_{rvp}	p_σ	v_θ	p_π	$\lambda = 0.5$	2	6	2890
α_{vp}	p_σ	v_θ	–	$\lambda = 0$	2	6	2177
α_{rp}	p_σ	–	p_π	$\lambda = 1$	8	0	2416
α_{rv}	$[p_\tau]$	v_θ	p_π	$\lambda = 0.5$	0	8	2077
α_v	$[p_\tau]$	v_θ	–	$\lambda = 0$	0	8	1655
α_r	$[p_\tau]$	–	p_π	$\lambda = 1$	0	0	1457
α_p	p_σ	–	–	–	0	0	1517

Evaluating positions using rollouts only (α_{rp}, α_r), value nets only (α_{vp}, α_v), or mixing both (α_{rvp}, α_{rv}); either using the policy network p_σ(α_{rvp}, α_{vp}, α_{rp}), or no policy network (α_{rvp}, α_{vp}, α_{rp}), that is, instead using the placeholder probabilities from the tree policy p_τ throughout. Each program used 5 s per move on a single machine with 48 CPUs and 8 GPUs. Elo ratings were computed by BayesElo.



有问题吗?

- 请随时举手提问。

BUSS 3620.人工智能导论

#2. 代码示例：井字游戏

刘佳璐

安泰经济与管理学院

上海交通大学

代码框架

迷宫

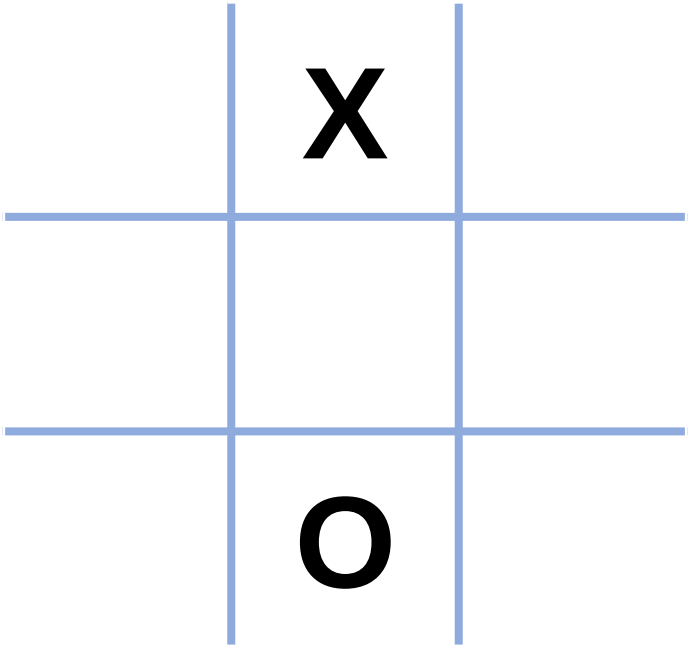
- 将问题抽象
  - 如何让计算机明白迷宫？
- 按照解搜索问题的步骤解迷宫
- 输出成果

井字游戏

- 将问题抽象
  - 如何让计算机明白井字游戏？
- 按照解对抗搜索问题的步骤制作游戏AI
- 输出成果

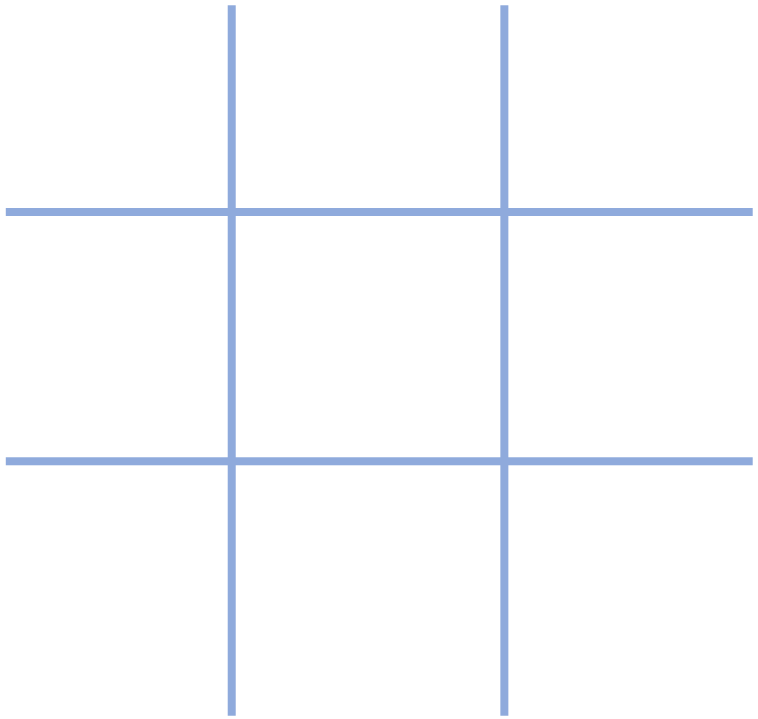
如何让计算机明白井字游戏?

- 游戏棋盘/状态s
  - 三个变量: X, O, Empty
  - 列表：[[row1], [row2], [row3]]
- 选手：X, O
- 行动a
  - 元组：(i, j)



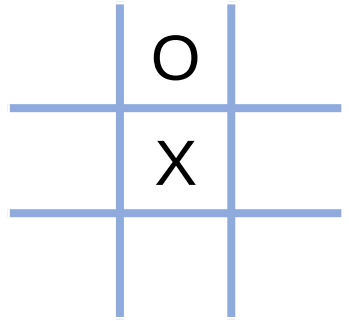
练习 #3

- 定义函数initial_state():
  - 返回游戏的初始状态S_0

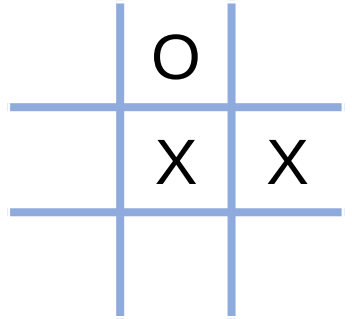


练习 #4

- 定义函数player(s):
  - 功能：返回状态s下可以行动的玩家
    - 假设X是先手玩家
  - 输入：一个状态
  - 输出：X or O
  - 提示：观察游戏中现有的O和X的数量

player(

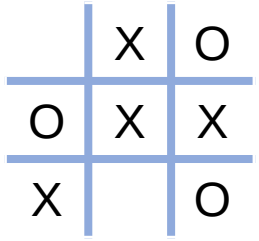
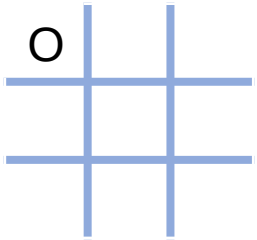
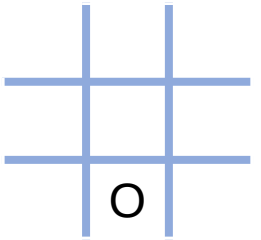
) = X

player(

) = O

练习 #5

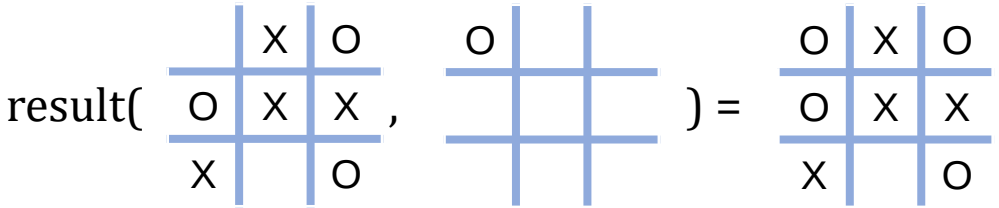
- 定义函数actions(s):
  - 功能：返回状态s下所有的合法行动集合
  - 输入：一个状态
  - 输出：一个行动集合

actions() = {  ,  }

练习 #6

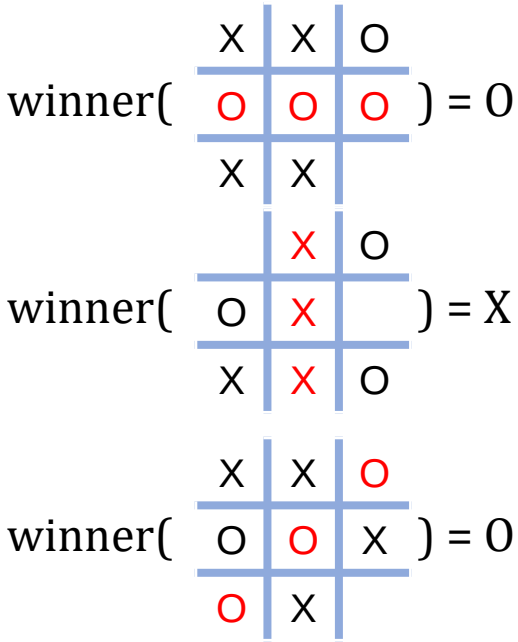
- 定义函数results(s, a):
  - 功能：返回在状态s下执行行动a后到达的新状态
    - 如果行动a不是合法行动，报错并退出程序
    - 不能改变原本的游戏棋盘
      - 提示：用包copy中的deepcopy
  - 输入：一个状态，一个行动
  - 输出：一个状态

```
>>> s=[1,2,3]
>>> s1=s
>>> s1[0]='a'
>>> s
['a', 2, 3]
```



练习 #7

- 定义函数winner(s):
  - 功能：返回状态s下的获胜方（如果存在）
    - 获胜条件：任意三个标记形成一条直线（横，竖，对角线）
  - 输入：一个状态
  - 输出：X or O or None



练习 #8

- 定义函数terminal(s):
  - 功能：确认状态s是否是结束状态(terminal state)
    - 结束状态
      - 有一方已经获胜，或
      - 棋盘上已经没有空格子
  - 输入：一个状态
  - 输出：True or False

terminal(O \| \|
 O \| X \|) = False
 X \| O \| X

terminal(O \| \| X
 O \| X \|) = True
 X \| O \| X

练习 #8

- 定义函数utility(s):
  - 功能：返回结束状态(terminal state) s的最终数值表示
    - 如果X选手胜利，为1
    - 如果O选手胜利，为-1
    - 其他为0
  - 输入：一个结束状态
  - 输出：1 or 0 or -1

utility(O | | X / O | X | / X | O | X) = 1

utility(O | X | X / X | O | / O | X | O) = -1

极小极大算法 Minimax

function Max-Value(*state*):

    if Terminal(*state*) is true:

        return Utility(*state*)

    v=-∞

    for *action* in Actions(*state*):

        v = max(v, Min-Value(Result(*state*, *action*)))

    return v

```
114    def Max_Value(board):
115
116        if terminal(board):
117            return utility(board)
118
119        v=-np.inf
120        action=actions(board)
121        for a in action:
122            v = max(v,Min_Value(result(board, a)))
123
124        return v
```

极小极大算法 Minimax

function Min-Value(*state*):

if Terminal(*state*) is true:

return Utility(*state*)

$v=\infty$

for *action* in Actions(*state*):

v = min(v, Max-Value(Result(*state*, *action*)))

return v

```
126  def Min_Value(board):
127      if terminal(board):
128          return utility(board)
129
130      action=actions(board)
131      v=np.inf
132
133      for a in action:
134          v = min(v, Max_Value(result(board, a)))
135
136      return v
```

极小极大算法 Minimax

- Minimax(s)函数：
  - 功能：返回状态 s下的最优的行动a
    - Max Player选择actions(s)中Min-Value(result(s, a))最大的一个行动a
    - Min Player选择actions(s)中Max-Value(result(s, a))最小的一个行动a
  - 输入：一个状态
  - 输出：一个行动

```python
def minimax(board):
    """
    Returns the optimal action for the current player on the board.
    """

    if terminal(board):
        return None

    action=actions(board)

    if player(board)==X:
        v=-np.inf
        for a in action:
            temp=Min_Value(result(board, a))
            if temp>v:
                v=temp
                solution=a
        return solution

    if player(board)==O:
        v=np.inf
        for a in action:
            temp=Max_Value(result(board, a))
            if temp<v:
                v=temp
                solution=a
        return solution
```

有问题吗?

- 请随时举手提问。