博弈算法

赵耀

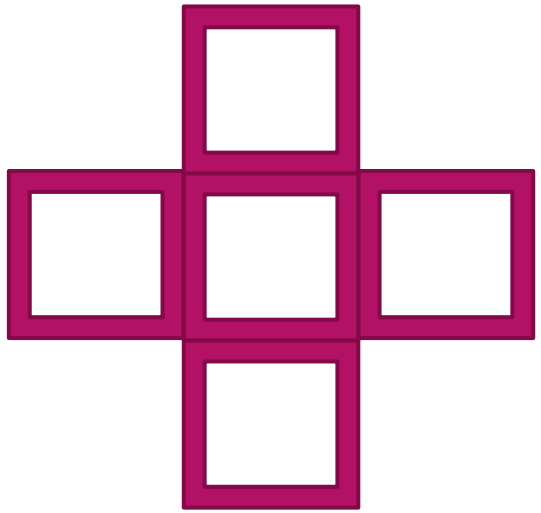
Minimax 算法

- 极大极小值搜索算法（Minimax Algorithm）是解决博弈树问题的基本方法

```
def minimax_decision(state, game):
    """Given a state in a game, calculate the best move by searching
    forward all the way to the terminal states. [Figure 5.3]"""

    player = game.to_move(state)

    def max_value(state):
        if game.terminal_test(state):
            return game.utility(state, player)
        v = -infinity
        for a in game.actions(state):
            v = max(v, min_value(game.result(state, a)))
        return v

    def min_value(state):
        if game.terminal_test(state):
            return game.utility(state, player)
        v = infinity
        for a in game.actions(state):
            v = min(v, max_value(game.result(state, a)))
        return v

    # Body of minimax_decision:
    return argmax(game.actions(state),
                  key=lambda a: min_value(game.result(state, a)))
```

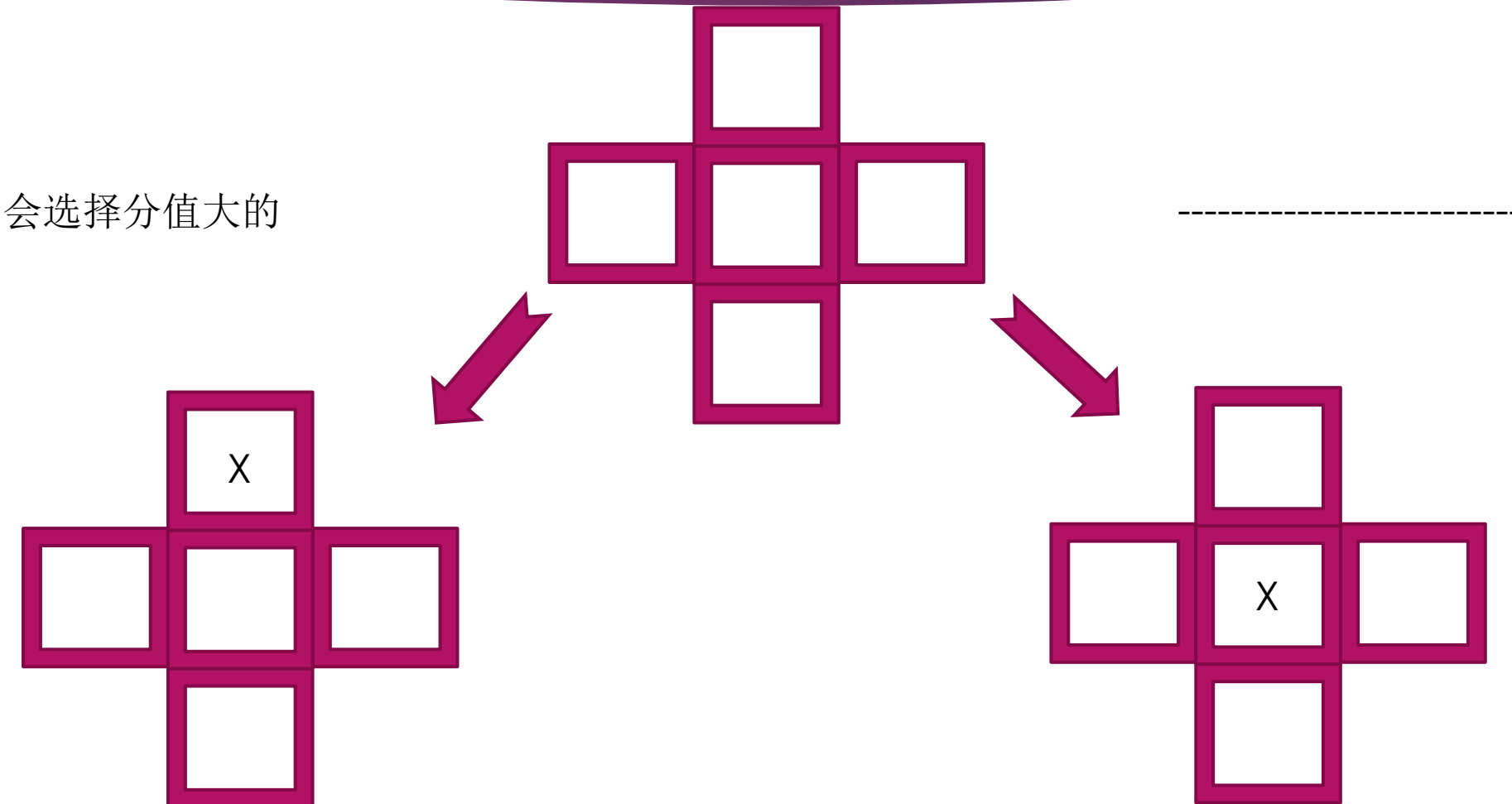
一个简单的例子

- 给右边的十字框填写X或者O
- X方先下
- 终止条件：
  - 如果连在一起的X或者O达到2个，则为胜利方
- utility函数：
  - 如果X胜利，分值为1；
  - 如果O胜利，分值为-1
  - 如果平局，分值为0。



一个简单的例子：第一层-MAX

MAX()
作为X想赢，会选择分值大的方案

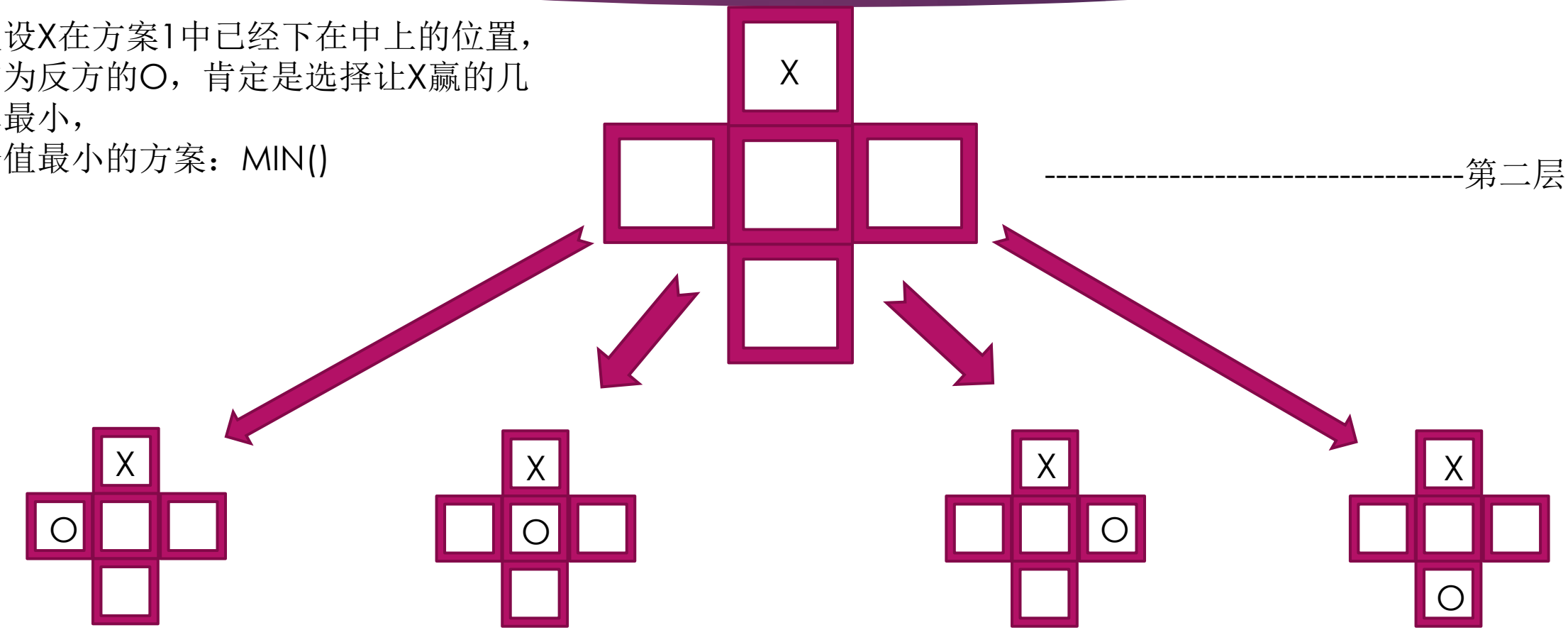


----------第一层

-----第二层

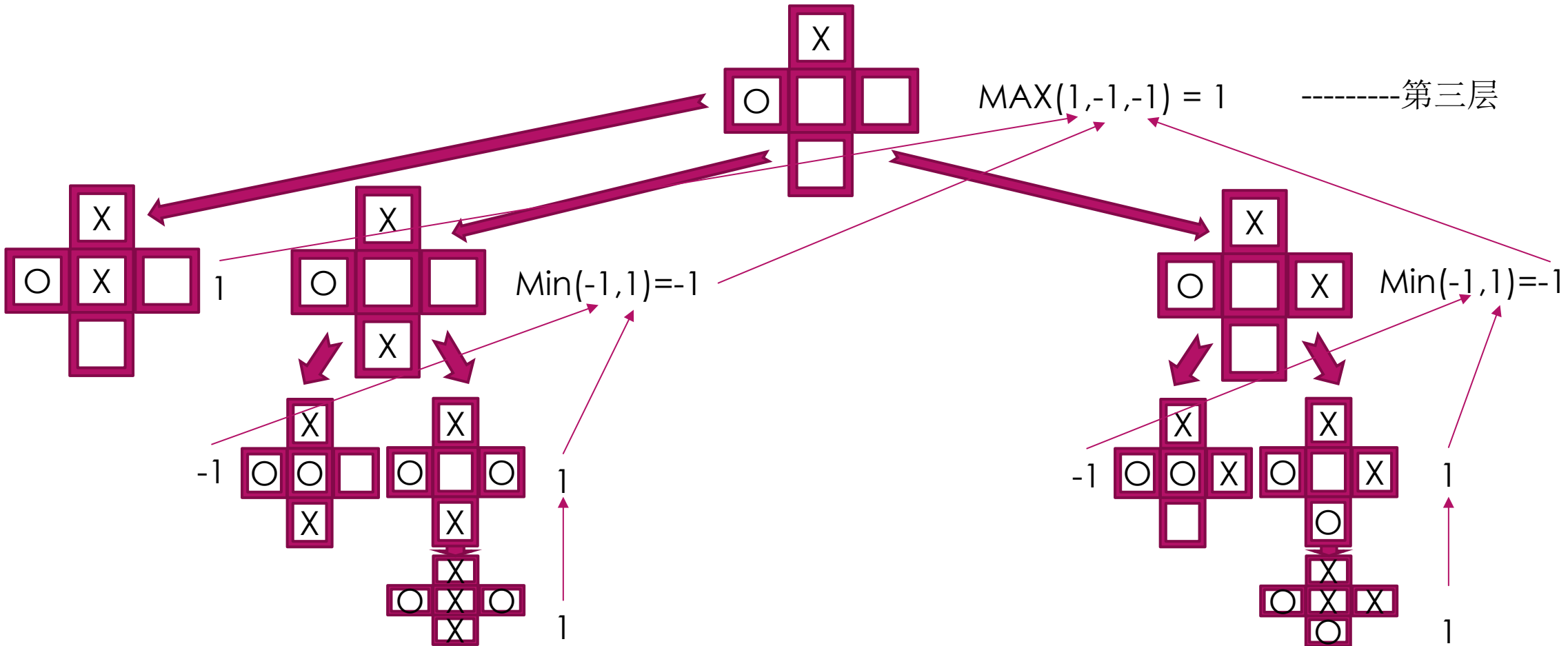
第二层：求MIN（）

假设X在方案1中已经下在中上的位置，
作为反方的O，肯定是选择让X赢的几率最小，
分值最小的方案：MIN()



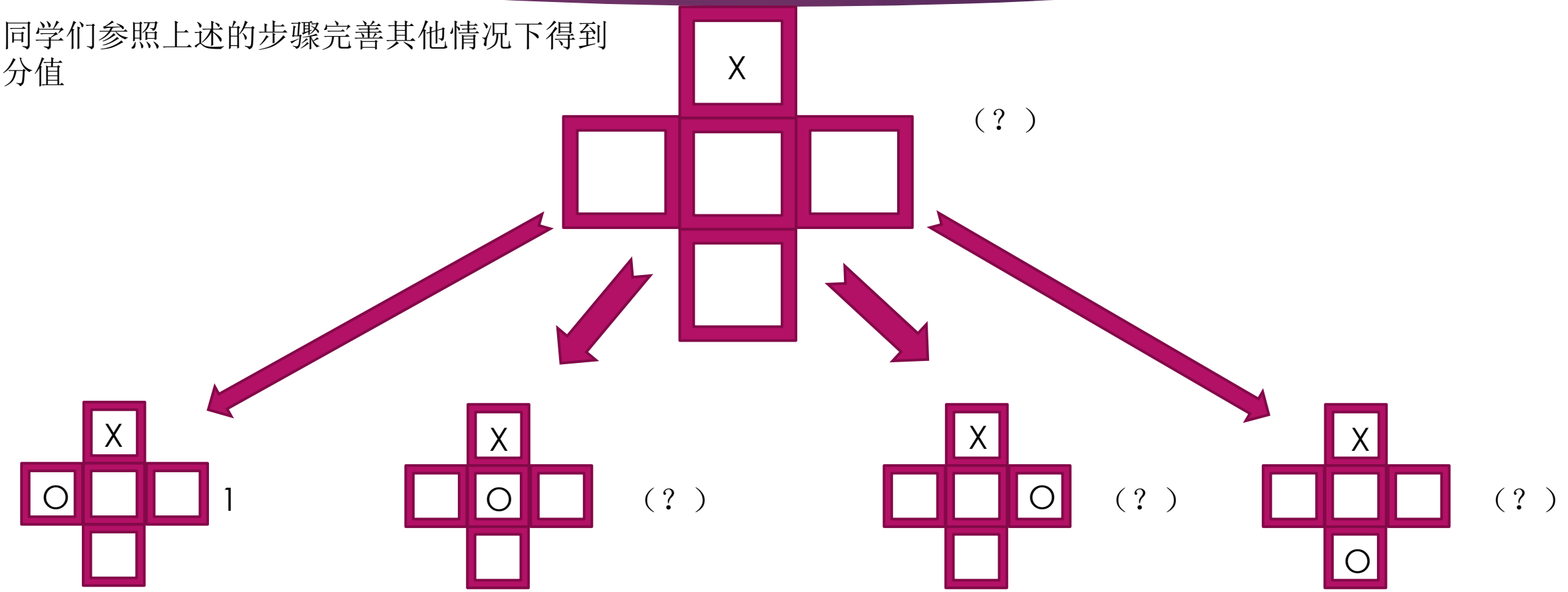
----------第二层

第三层到第六层

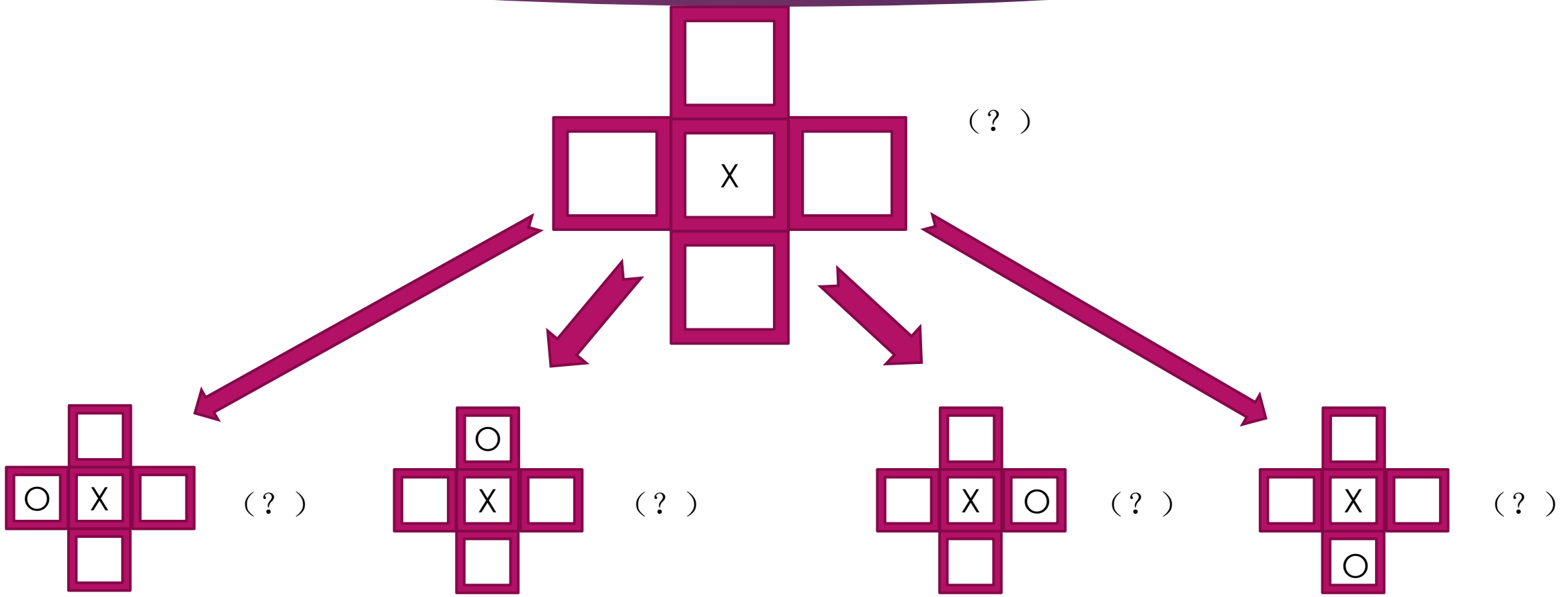


完善并填空

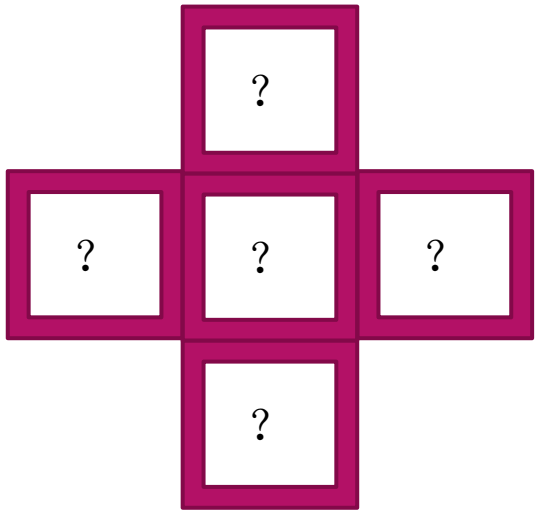
请同学们参照上述的步骤完善其他情况下得到的分值



完善并填空



最终X的第一步会下在哪呢？



在上述执行MiniMax算法中

- 如果是对称的情况，是否有必要搜索2遍？
- 在这个过程中，是否可以总结出哪些是冗余的搜索？
- 如果让你设计一个合适的评估函数，你会怎么设计？

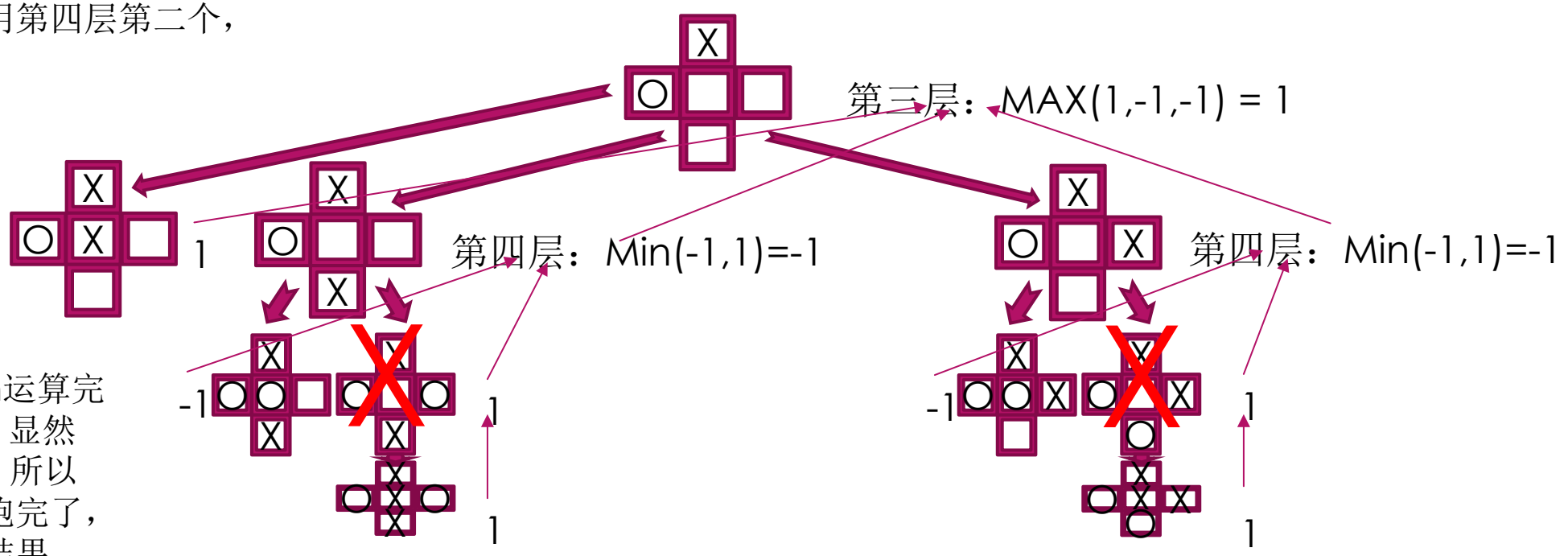
Alpha-Beta 剪枝算法

见右图，第三层需要得到第四层的最大值，随着
搜索的进行，第四层最左边已经得到了值1，并且将1返回给了
第三层，此时第三层继续调用第四层第二个，
第四层第二个完成了左分支，
得到了-1，注意
这个-1比1要小。

思考，此时，第四层第二个
是否需要继续其右
分支？

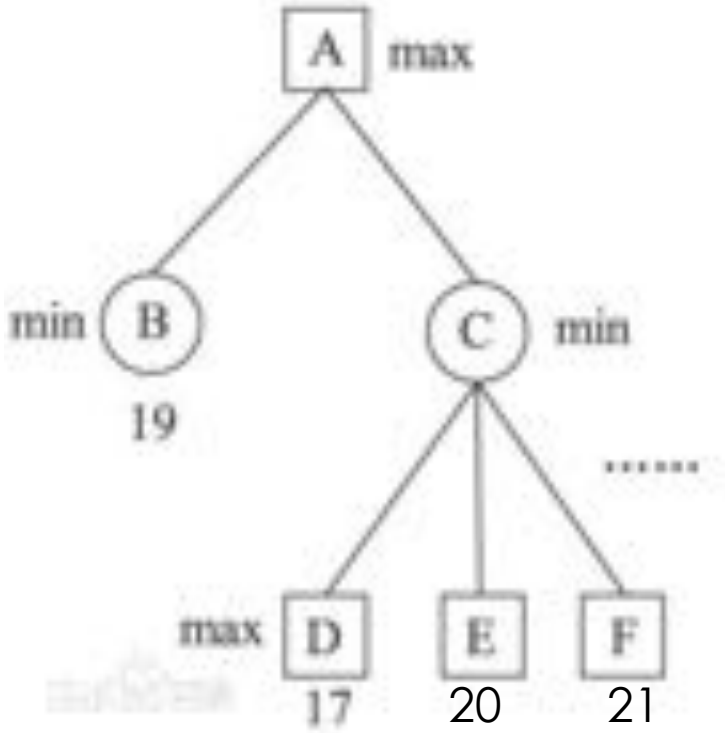
如果右分支比-1大，显然Min运算完
还是-1；如果右分支比-1小，显然
第三层的MAX仍然是选择1。所以
第四层第二个的右分支就算跑完了，
也绝不会影响第三层的最终结果。

结论：第四层第二个的右分支可以剪掉



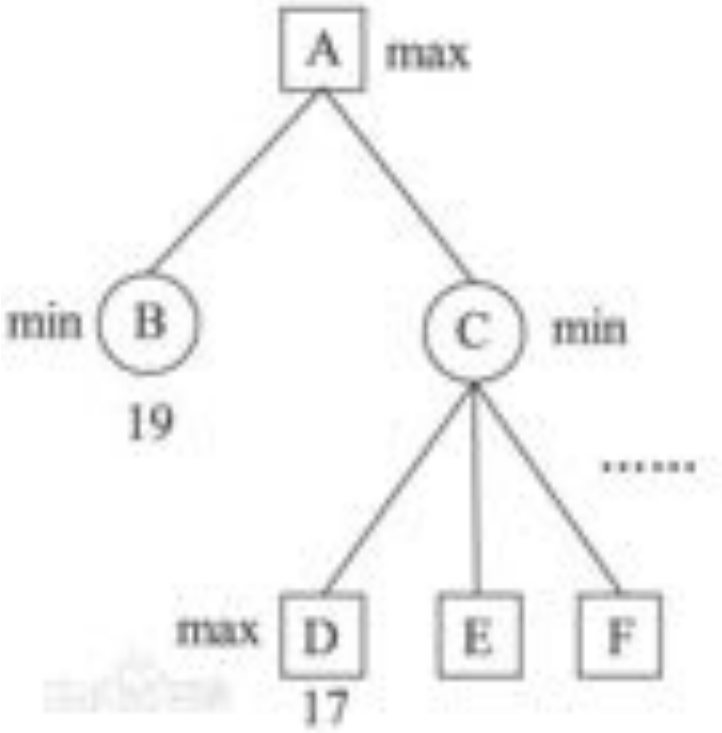
Alpha剪枝

- 节点A 的值应是节点 B 和节点C 的值中之较大者。现在已知节点 B 的值大于节点D 的值。由于节点C 的值应是它的诸子节点的值中之极小者，此极小值一定小于等于节点D 的值，因此亦一定小于节点B 的值，这表明，继续搜索节点C 的其他诸子节点E，F已没有意义，它们不能做任何贡献，于是把以节点 C 为根的子树全部剪去。这种优化称为Alpha 剪枝。
- 思考：如果搜索的时候，分支E和F在D前面？



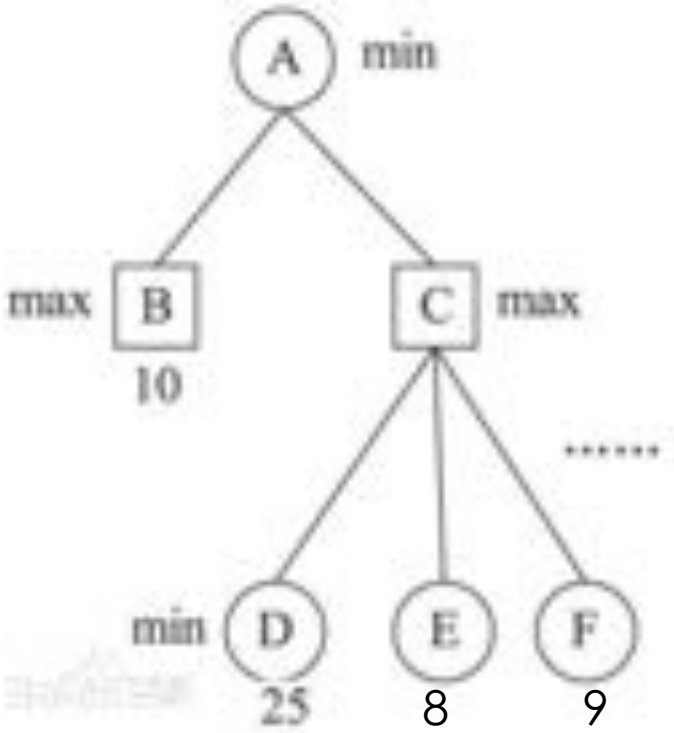
Alpha剪枝解析

- MAX层：在节点A，用alpha保存当前在子节点中找到的最大值，这个alpha值会随着函数调用传递到下一层
- 下一层为MIN层，MIN层的节点当前找到的最小值如果<=alpha值，那么没必要再继续寻找，终止
- 一个min节点，需要保持更新自己的beta值，所以如果该节点分支没终止，而且当前找到的最小值<beta，需要更新beta值，这个beta值会传递到这个节点的下一层



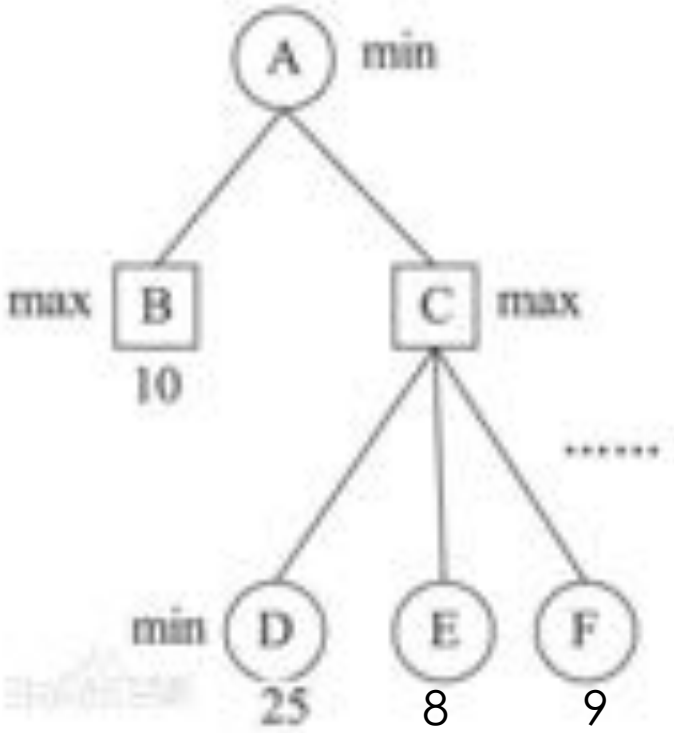
Beta剪枝

- 与极大值冗余对偶的现象，称为极小值冗余。节点A 的值应是节点B 和节点C 的值中之较小者。 现在已知节点B 的值小于节点D 的值。 由于节点C 的值应是它的诸子节点的值中之极大者，此极大值一定大于等于节点D 的值,因此也大于节点B的值，这表明，继续搜索节点C 的其他诸子节点已没有意义，并可以把以节点C 为根的子树全部剪去，这种优化称为Beta剪枝。
- 思考一下，如果E和F的分支在D前面？



Beta剪枝解析

- MIN层：在节点A，用beta保存当前在子节点中找到的最小值，这个beta值会随着函数调用传递到下一层
- 下一层为MAX层，MAX层的节点当前找到的最小值如果>=beta值，那么没必要再继续寻找，终止
- 一个max节点，需要保持更新自己的alpha值，所以如果该节点分支没终止，而且当前找到的最大值>alpha，需要更新alpha值，这个alpha值会传递到这个节点的下一层



Alpha-Beta 剪枝算法

- 把Alpha -Beta 剪枝应用到极大极小算法中，就形成了Alpha -Beta 搜索算法
- 它的优化来自于minimax的特性，并不会改变minimax的结果
- 优化的程度与节点的先后顺序相关

```
44  def alphabeta_search(state, game):
45      """Search game to determine best action; use alpha-beta pruning.
46      As in [Figure 5.7], this version searches all the way to the leaves."""
47
48      player = game.to_move(state)
49
50      # Functions used by alphabeta
51      def max_value(state, alpha, beta):
52          if game.terminal_test(state):
53              return game.utility(state, player)
54          v = -infinity
55          for a in game.actions(state):
56              v = max(v, min_value(game.result(state, a), alpha, beta))
57              if v >= beta:
58                  return v
59              alpha = max(alpha, v)
60          return v
61
62      def min_value(state, alpha, beta):
63          if game.terminal_test(state):
64              return game.utility(state, player)
65          v = infinity
66          for a in game.actions(state):
67              v = min(v, max_value(game.result(state, a), alpha, beta))
68              if v <= alpha:
69                  return v
70              beta = min(beta, v)
71          return v
72
73      # Body of alphabeta_cutoff_search:
74      best_score = -infinity
75      beta = infinity
76      best_action = None
77      for a in game.actions(state):
78          v = min_value(game.result(state, a), best_score, beta)
79          if v > best_score:
80              best_score = v
81              best_action = a
82      return best_action
```

beta剪枝

更新alpha值

alpha剪枝

更新beta值

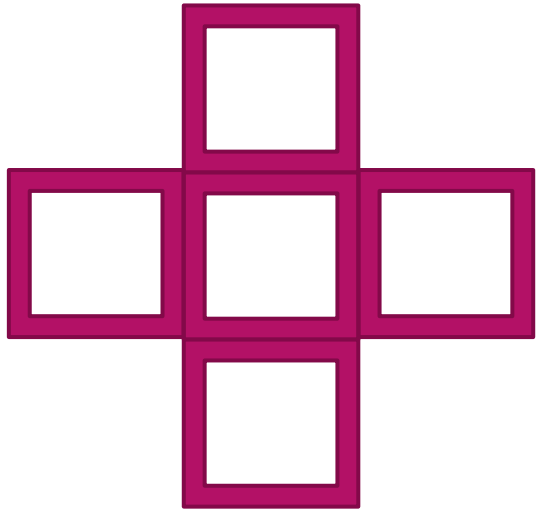
H-Minimax

- 使用eval函数取代utility函数
- 使用cutoff_test取代terminal_test
- 思考：这样做的好处是什么？

```
def alphabeta_cutoff_search(state, game, d=4, cutoff_test=None, eval_fn=None):
    """Search game to determine best action; use alpha-beta pruning.
    This version cuts off search and uses an evaluation function."""

    player = game.to_move(state)

    # Functions used by alphabeta
    def max_value(state, alpha, beta, depth):
        if cutoff_test(state, depth):
            return eval_fn(state)
        v = -infinity
        for a in game.actions(state):
            v = max(v, min_value(game.result(state, a),
                                 alpha, beta, depth + 1))
            if v >= beta:
                return v
            alpha = max(alpha, v)
        return v

    def min_value(state, alpha, beta, depth):
        if cutoff_test(state, depth):
            return eval_fn(state)
        v = infinity
        for a in game.actions(state):
            v = min(v, max_value(game.result(state, a),
                                 alpha, beta, depth + 1))
            if v <= alpha:
                return v
            beta = min(beta, v)
        return v

    # Body of alphabeta_cutoff_search starts here:
    # The default test cuts off at depth d or at a terminal state
    cutoff_test = (cutoff_test or
                   (lambda state, depth: depth > d or
                    game.terminal_test(state)))
    eval_fn = eval_fn or (lambda state: game.utility(state, player))
    best_score = -infinity
    beta = infinity
    best_action = None
    for a in game.actions(state):
        v = min_value(game.result(state, a), best_score, beta, 1)
        if v > best_score:
            best_score = v
            best_action = a
    return best_action
```

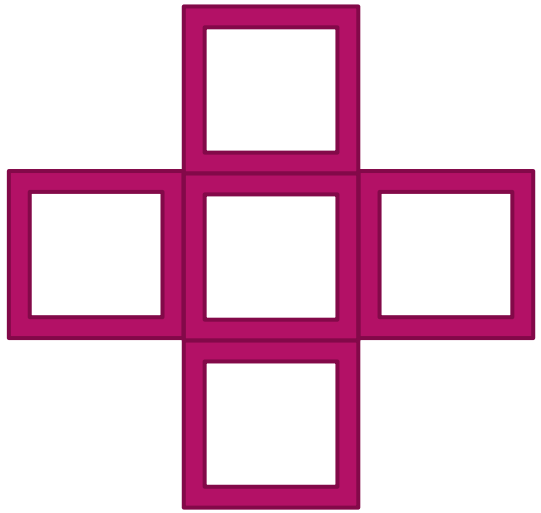
估值函数的设计

- 怎样设计这个游戏的估值函数？



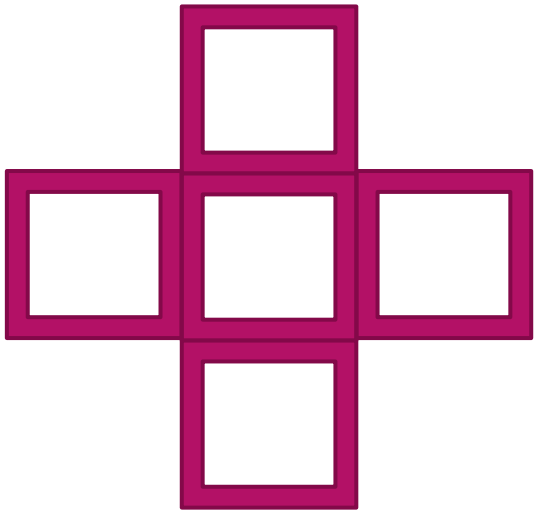
估值函数的设计

- 怎样设计这个游戏的估值函数？
- 可以这样设计：
  - 如果X处于中间的位置，分值为4，处于四边的位置，分值为1
  - 如果O处于中间的位置，分值为-4，处于四边的位置，分值为-1
  - 按上述累加所有X和O的分值



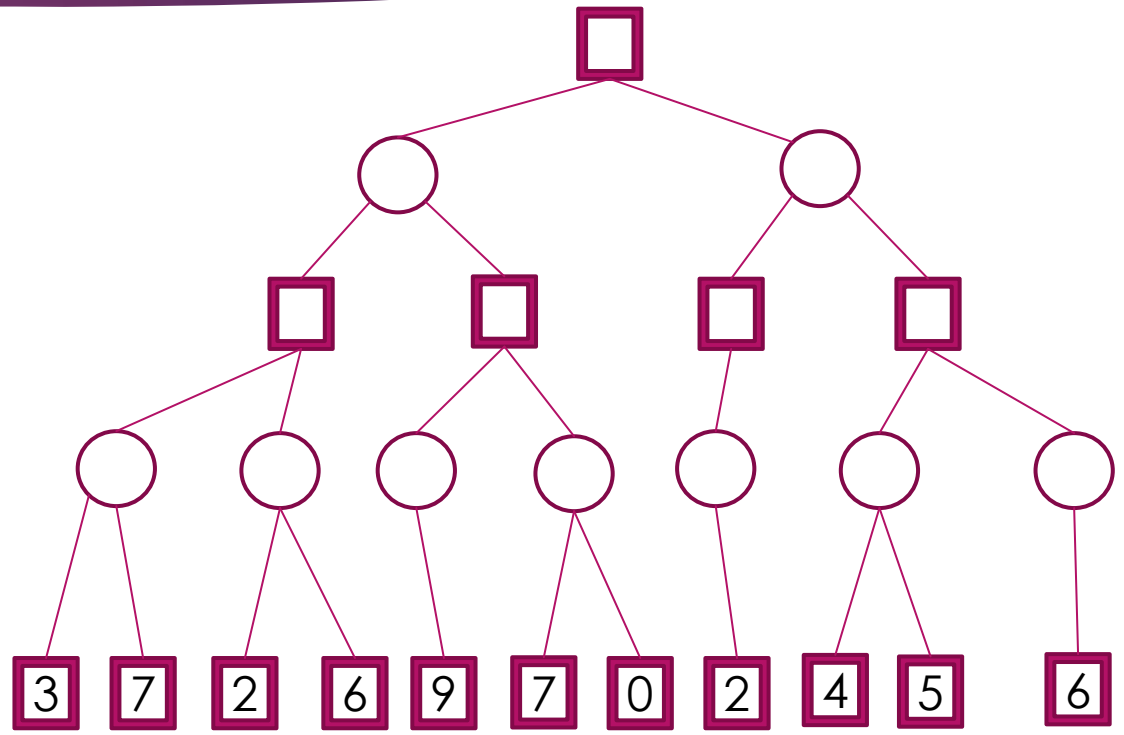
应用Alpha-Beta 剪枝算法

- 用Alpha-Beta剪枝算法对右图的游戏进行剪枝



应用Alpha-Beta 剪枝算法

- 一次MINIMAX算法执行结果如右图，用Alpha-Beta剪枝算法对右图进行剪枝



Tic-Tac-Toe

- http://aimacode.github.io/aima-javascript/5-Adversarial-Search/