搜索求解2

主讲： 王亚星、刘夏雷、郭春乐

南开大学计算机学院

致谢：本课件主要内容来自浙江大学吴飞教授、

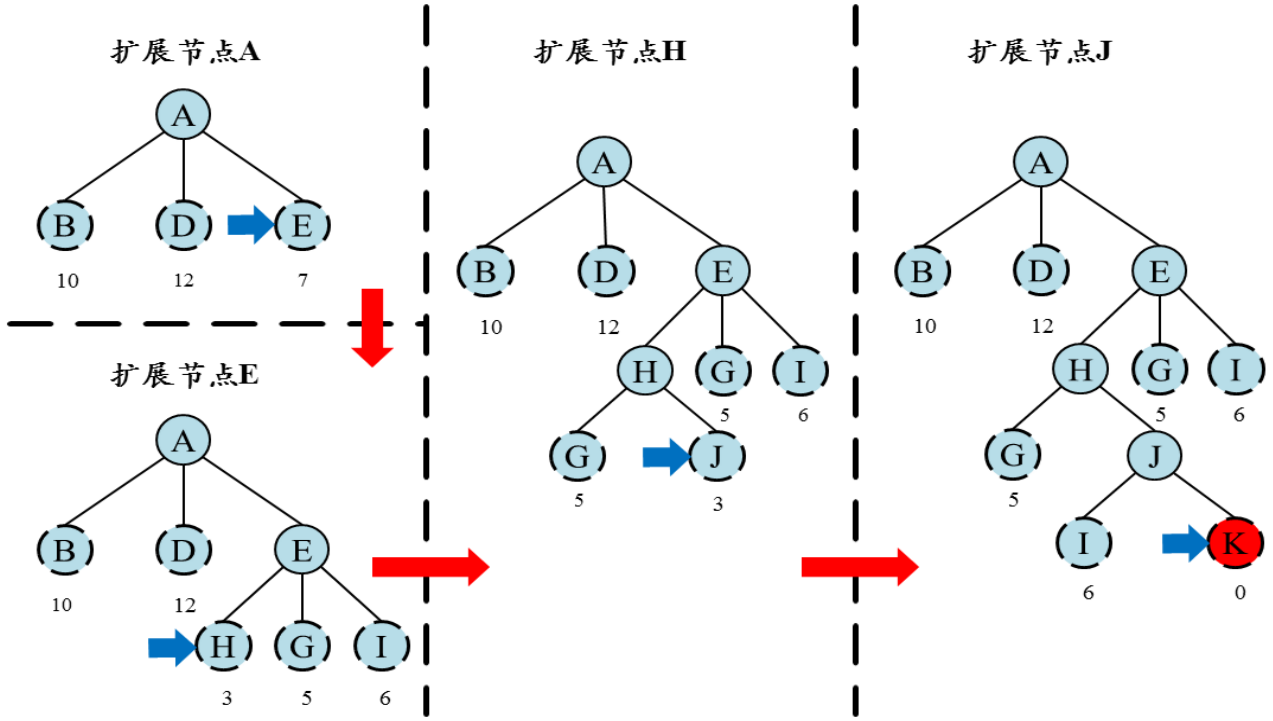
南开大学程明明教授

提纲

- 搜索算法基础
- 启发式搜索
- 对抗搜索
- 蒙特卡洛树搜索

内容回顾：贪婪最佳优先搜索

- 贪婪最佳优先搜索：评价函数$f(n)$=启发函数$h(n)$
  - 例：启发函数为任意一个城市与终点城市K之间的直线距离



贪婪最佳优先搜索的过程

算法找到了一条从起始结点到终点结点的路径A → E → H → J → K，但这条路径并不是最短路径，实际上最短路径为A → E → G → K。

内容回顾：A*算法

- **评价函数：$\mathbf{f}(\boldsymbol{n}) = \mathbf{g}(\boldsymbol{n}) + \boldsymbol{h}(\boldsymbol{n})$**
  - $\mathbf{g}(\boldsymbol{n})$表示从起始结点到结点$\boldsymbol{n}$的开销代价值
  - $\boldsymbol{h}(\boldsymbol{n})$表示从结点$\boldsymbol{n}$到目标结点路径中所估算的最小开销代价值
  - $\mathbf{f}(\boldsymbol{n})$可视为经过结点$\boldsymbol{n}$、具有最小开销代价值的路径。

$$\underbrace{\boldsymbol{f}(\boldsymbol{n})}_{\text{评价函数}} = \underbrace{\boldsymbol{g}(\boldsymbol{n})}_{\substack{\text{起始结点到结点}\boldsymbol{n}\text{代价} \\ \text{(当前最小代价)}}} + \underbrace{\boldsymbol{h}(\boldsymbol{n})}_{\substack{\text{结点}\boldsymbol{n}\text{到目标结点代价} \\ \text{(后续估计最小代价)}}}$$

单选题 1分

设置

1.假如可以对围棋的规则做出如下修改,其中哪个修改方案不影响使用本章介绍的Minimax算法求解该问题?( )

确定的、全局可观察的、
竞争对手轮流行动、
零和游戏(zero-sum)

A 由双方轮流落子,改为黑方连落两子后白方落一子。

B 双方互相不知道对方落子的位置。

C 由两人对弈改为三人对弈。

D 终局时黑方所占的每目(即每个交叉点)计1分,且事先给定了白方在棋盘上每个位置取得一目所获取的分数,假设这些分数各不相同。双方都以取得最高得分为目标。

提交

主观题 10分

设置

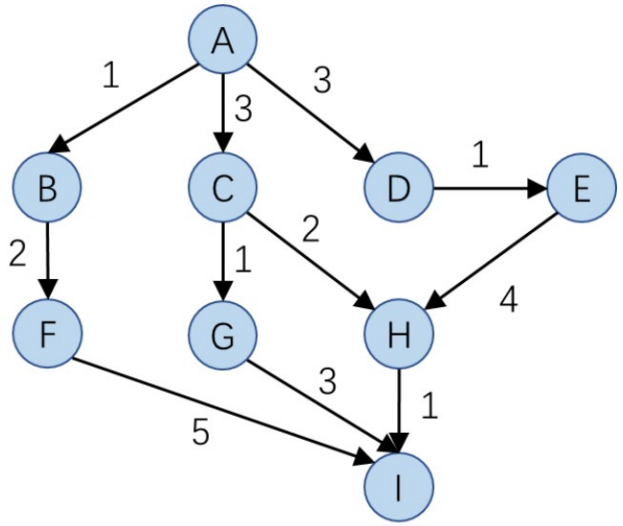


图1 状态转移图

1. 考虑图1中的问题，给定每个状态的启发函数如下表所示。若仍以状态A为初始状态、状态I为终止状态，请分别使用以下算法求解从A到I的路径，请画出算法找到第一条路径时的搜索树，并在搜索书中标出结点的扩展顺序，以及找到的路径。（若有多个节点拥有相同的扩展优先度，则优先扩展对应路径字典序较小的节点）。

(1) 基于树搜索的贪婪最佳优先搜索。

(2) 基于图搜索的A*算法。

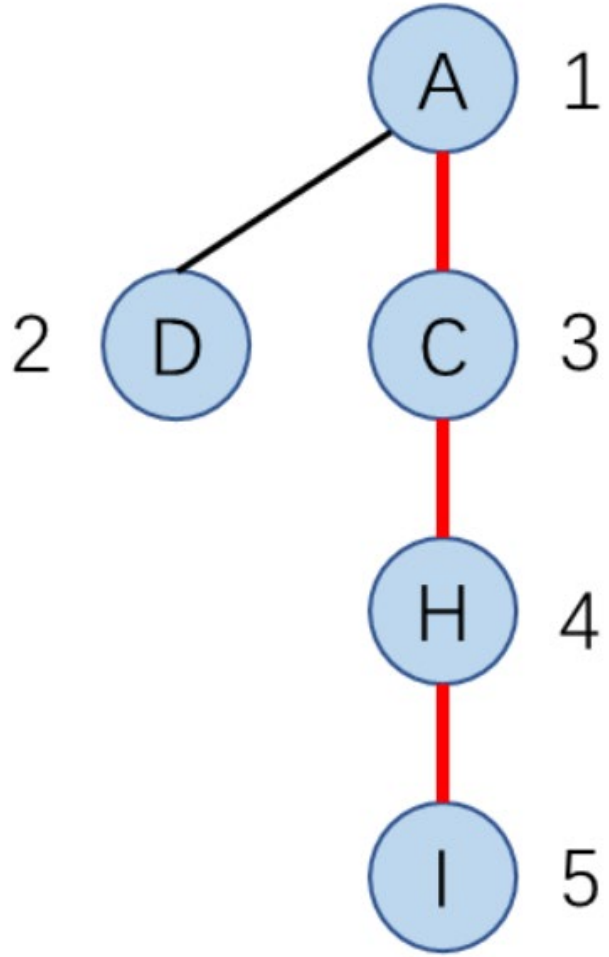
状态	A	B	C	D	E	F	G	H	I
启发函数	5	4	3	2	5	5	2	1	0

正常使用主观题需2.0以上版本雨课堂

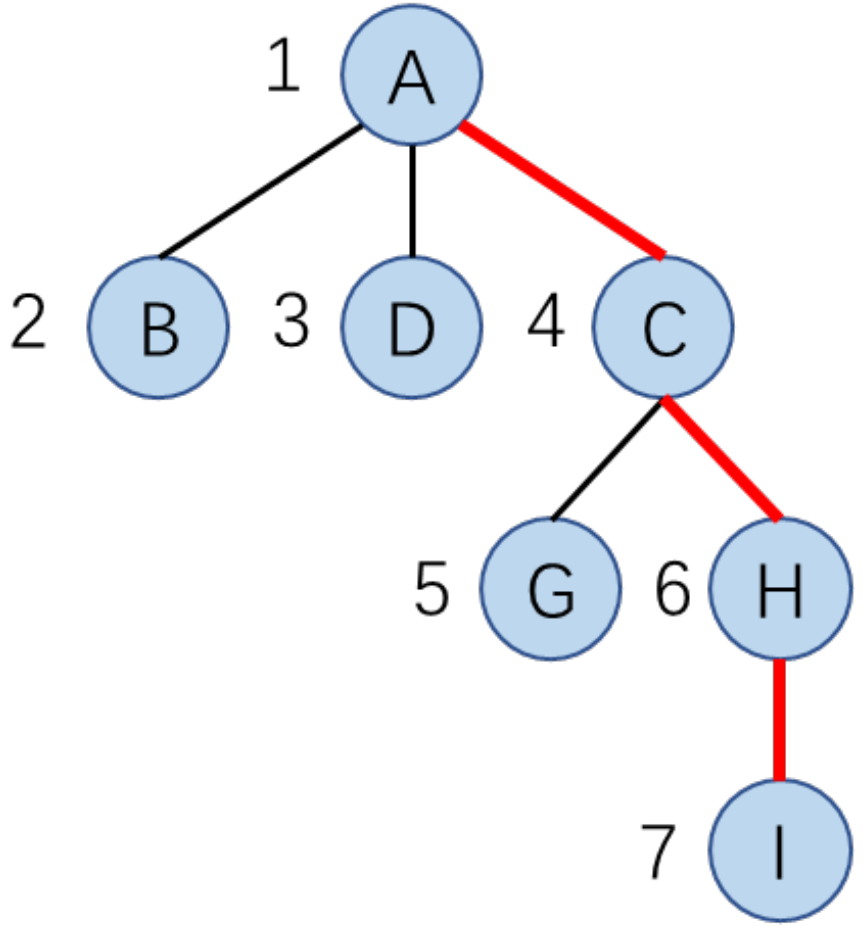
作答

习题答案

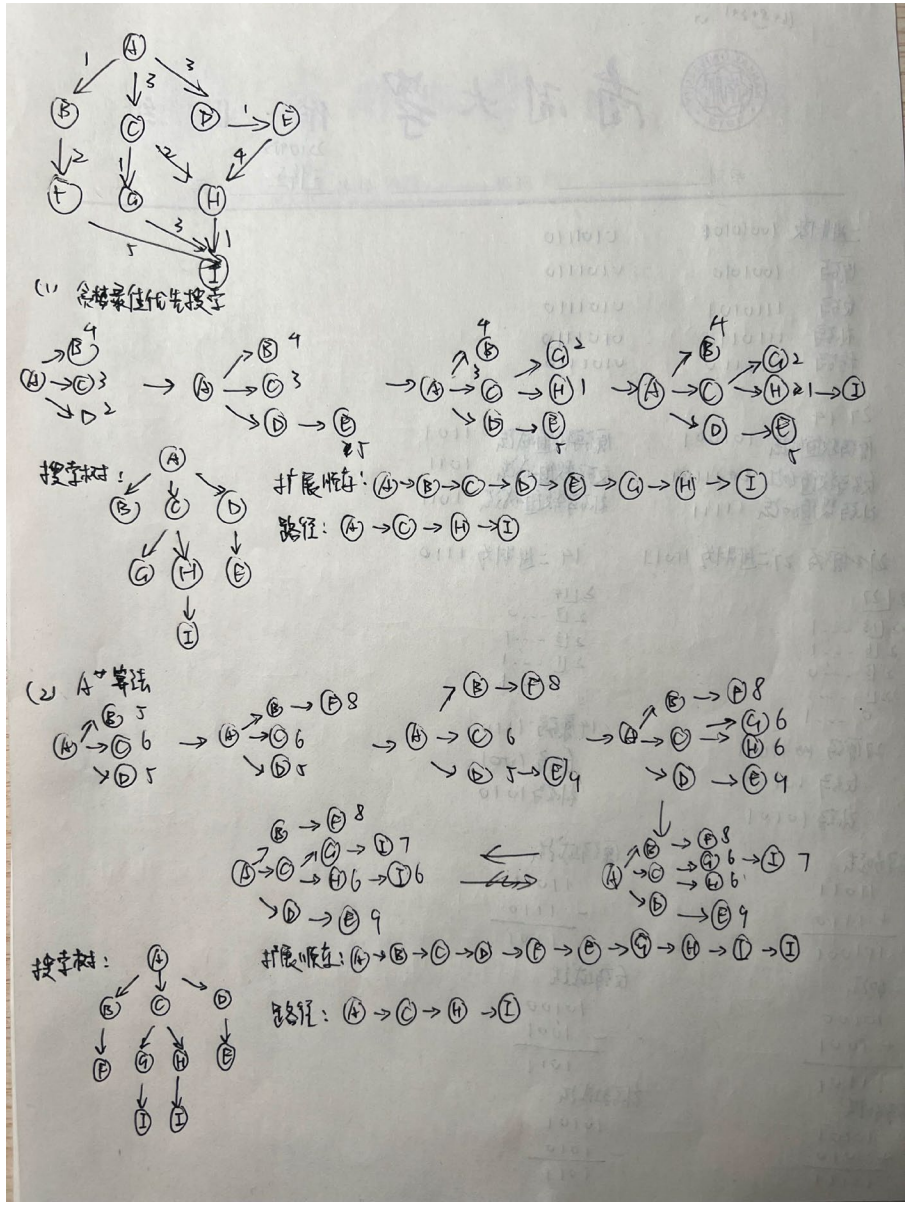
(1) 基于树搜索的贪婪最佳优先搜索。



(2) 基于图搜索的A*算法。



习题答案



王梓丞

对抗搜索

- 对抗搜索(Adversarial Search)也称为博弈搜索(Game Search)
- 在一个竞争的环境中，智能体(agents)之间通过竞争实现相反的利益，一方**最大化**这个利益，另外一方**最小化**这个利益。

对抗搜索：主要内容

- **最小最大搜索(Minimax Search)**
  - 最小最大搜索是在对抗搜索中最为基本的一种让玩家来计算最优策略的方法
- **Alpha-Beta剪枝搜索(Pruning Search)**
  - 一种对最小最大搜索进行改进的算法，即在搜索过程中可剪除无需搜索的分支节点，且不影响搜索结果。
- **蒙特卡洛树搜索(Monte-Carlo Tree Search)**
  - 通过采样而非穷举方法来实现搜索。

对抗搜索

- **本课程目前主要讨论在确定的、全局可观察的、竞争对手轮流行动、零和游戏(zero-sum)下的对抗搜索**
  - 所谓零和博弈是博弈论的一个概念，属非合作博弈。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”，双方不存在合作的可能。与“零和”对应，“双赢博弈”的基本理论就是“利己”不“损人”，通过谈判、合作达到皆大欢喜的结果。

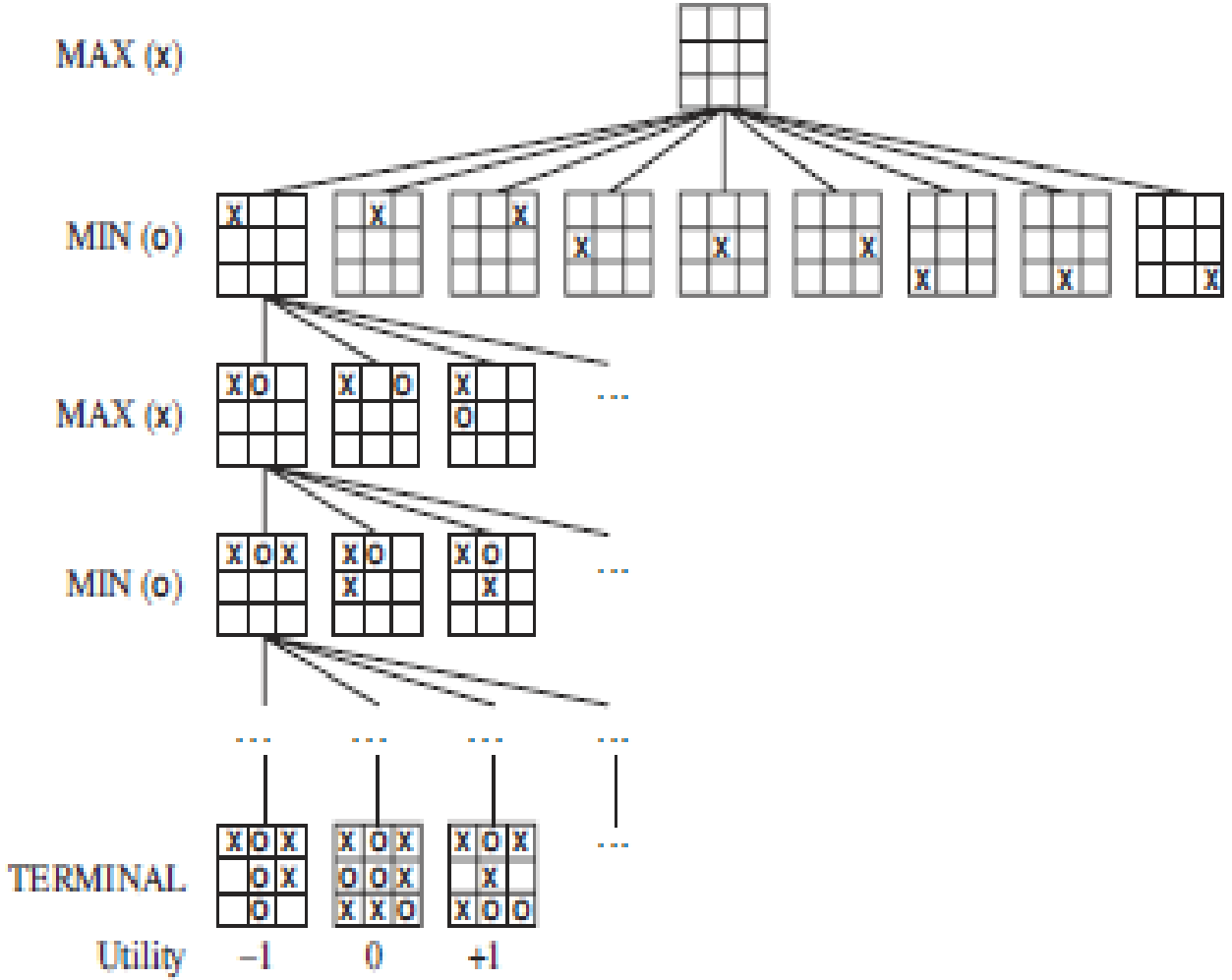
对抗搜索

- 本课程目前主要讨论在确定的、全局可观察的、竞争对手轮流行动、零和游戏(zero-sum)下的对抗搜索
- 两人对决游戏 (MAX and MIN, MAX先走) 可如下形式化描述，从而将其转换为对抗搜索问题

初始状态S_0	游戏所处于的初始状态
玩家$PLAYER(s)$	在当前状态s下，该由哪个玩家采取行动
行动$ACTIONS\ (s)$	在当前状态s下所采取的可能移动
状态转移模型$RESULT\ (s,a)$	在当前状态s下采取行动a后得到的结果
终局状态检测$TERMINAL-TEST\ (s)$	检测游戏在状态s是否结束
终局得分$UTILITY\ (s,p)$	在终局状态s时，玩家p的得分。

对抗搜索

- **Tic-Tac-Toe游戏的对抗搜索**
  - MAX先行，可在初始状态的9个空格中任意放一个X
  - MAX希望游戏终局得分高、MIN希望游戏终局得分低
  - 所形成游戏树的叶子结点有$9! = 362{,}880$,国际象棋的叶子节点数为10^{40}



Tic-Tac-Toe中部分搜索树

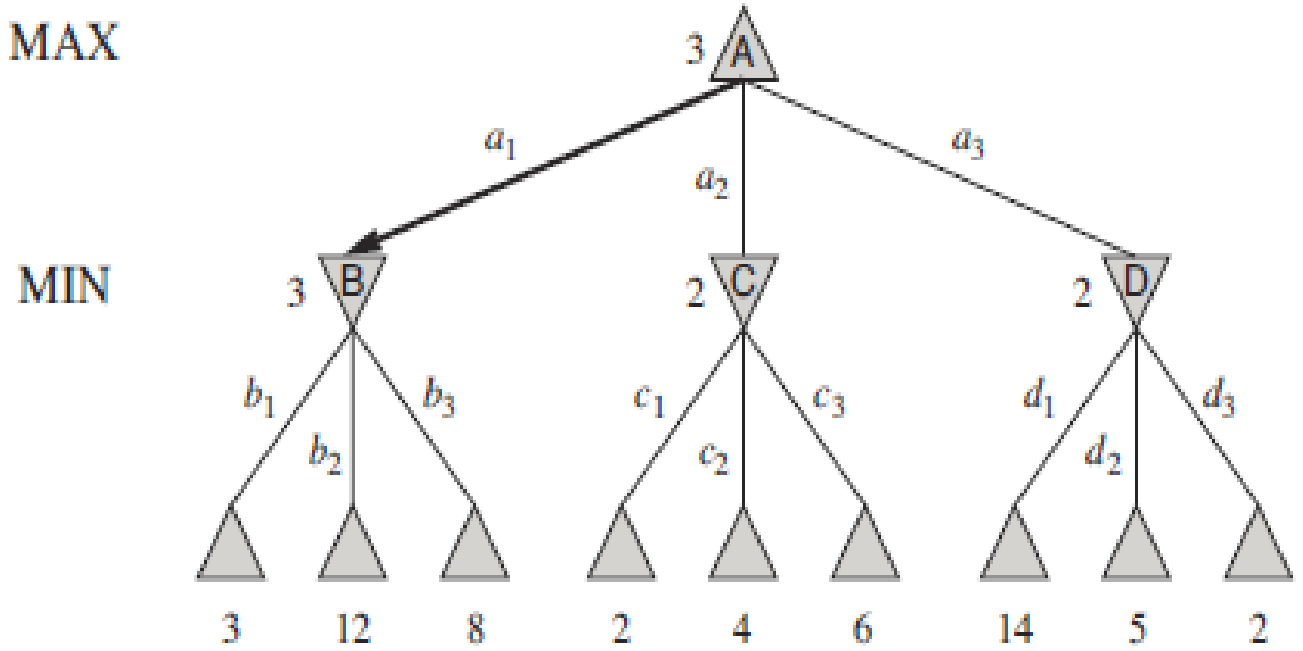
对抗搜索：minimax算法

- **给定一个游戏搜索树，minimax算法通过每个节点的minimax值来决定最优策略。**
  - MAX希望最大化minimax值，而MIN则相反

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

对抗搜索：minimax算法

- **给定一个游戏搜索树，minimax算法通过每个节点的minimax值来决定最优策略。**
  - 通过minimax算法，我们知道，对于MAX而言采取a_1行动是最佳选择，因为这能够得到最大minimax值(收益最大)。



对抗搜索：minimax算法

- **给定一个游戏搜索树，minimax算法通过每个节点的minimax值来决定最优策略。**
  - 通过minimax算法，我们知道，对于MAX而言采取a_1行动是最佳选择，因为这能够得到最大minimax值(收益最大)。

对抗搜索：minimax算法

- **Complete ?**
  - Yes (if tree is finite)
- **Optimal?**
  - Yes (against an optimal opponent)
- **Time complexity ?**
  - $O(b^m)$
  - m 是游戏树的最大深度
  - 在每个节点存在b个有效走法
- **Space complexity ?**
  - $O(b \times m)$ (depth-first exploration)

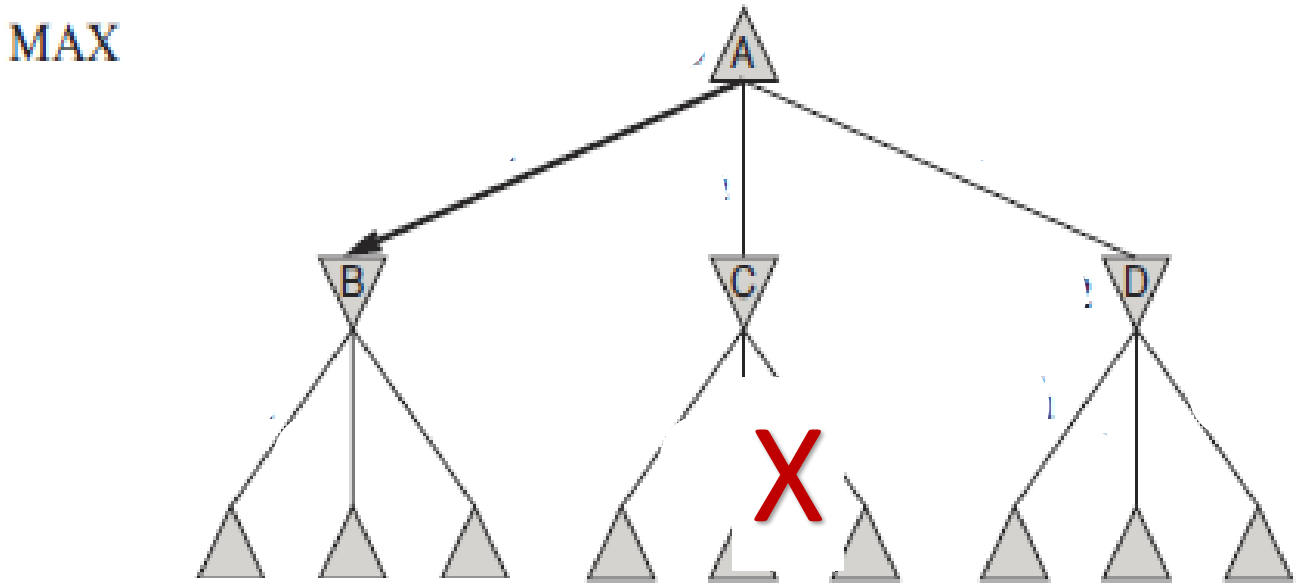
For chess, b ≈ 35, m ≈100
for "reasonable" games
→ exact solution
completely infeasible

对抗搜索：minimax算法

- **优点:**
  - 算法是一种简单有效的对抗搜索手段
  - 在对手也“尽力而为”前提下，算法可返回最优结果
- **缺点:**
  - 如果搜索树极大，则无法在有效时间内返回结果
- **改善:**
  - 使用alpha-beta pruning算法来减少搜索节点
  - 对节点进行采样、而非逐一搜索 (i.e., MCTS)

对抗搜索： Alpha-Beta 剪枝搜索

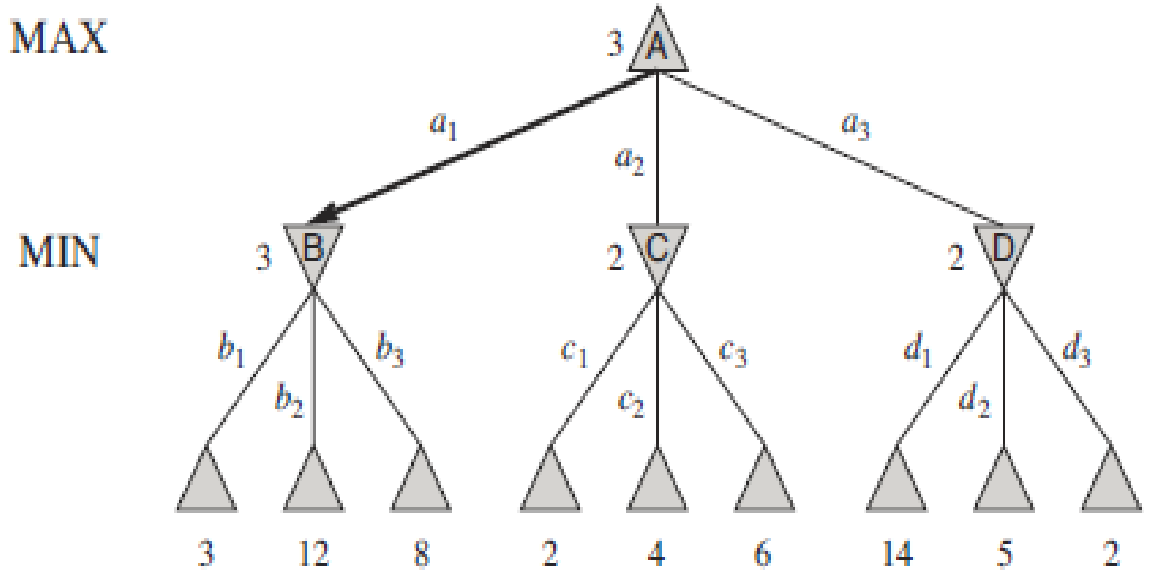
动机：



对抗搜索：Alpha-Beta 剪枝搜索

- 在极小化极大算法(minimax算法)中减少所搜索的搜索树节点数。该算法和极小化极大算法所得结论相同，但剪去了不影响最终结果的搜索分枝。

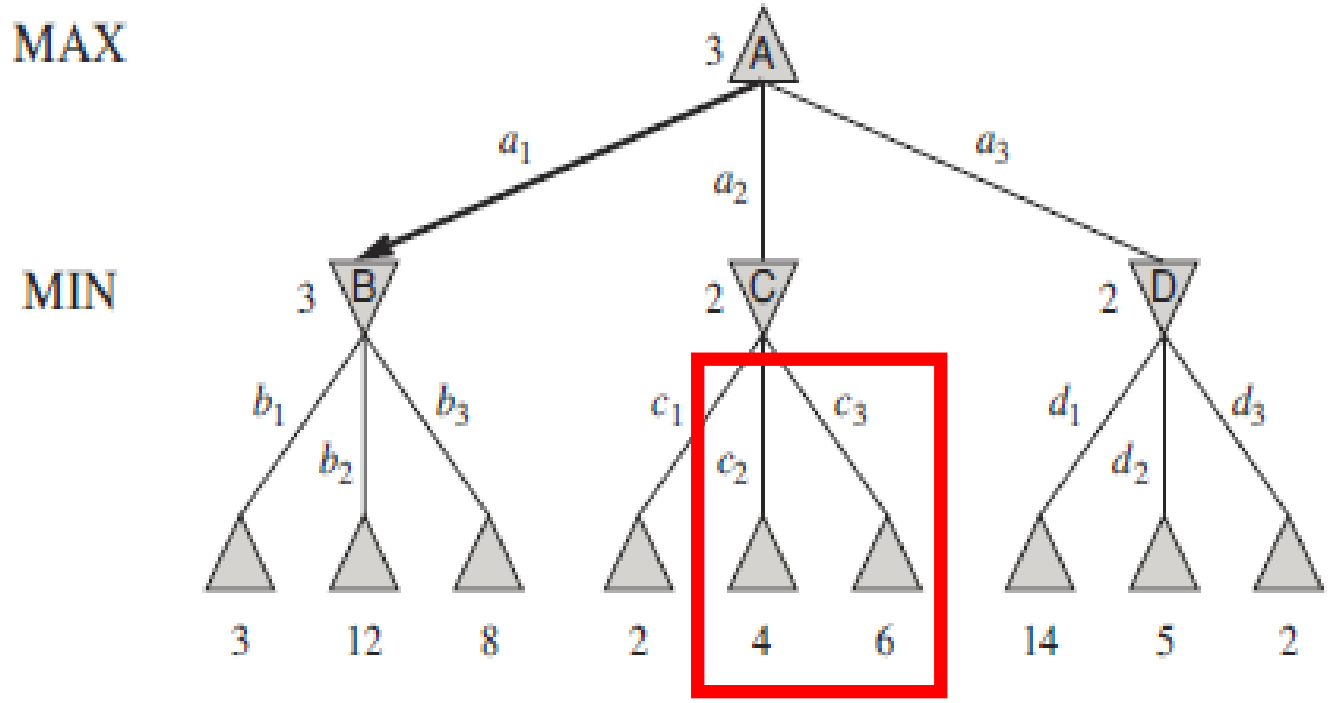
$MINIMAX(root)$



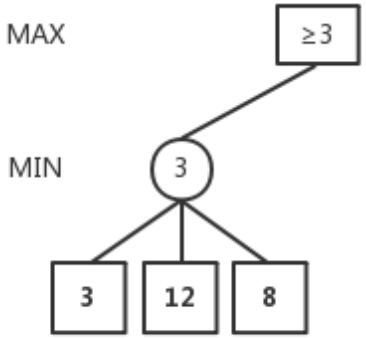
对抗搜索： Alpha-Beta 剪枝搜索

- 剪去不影响最终结果的搜索分枝。

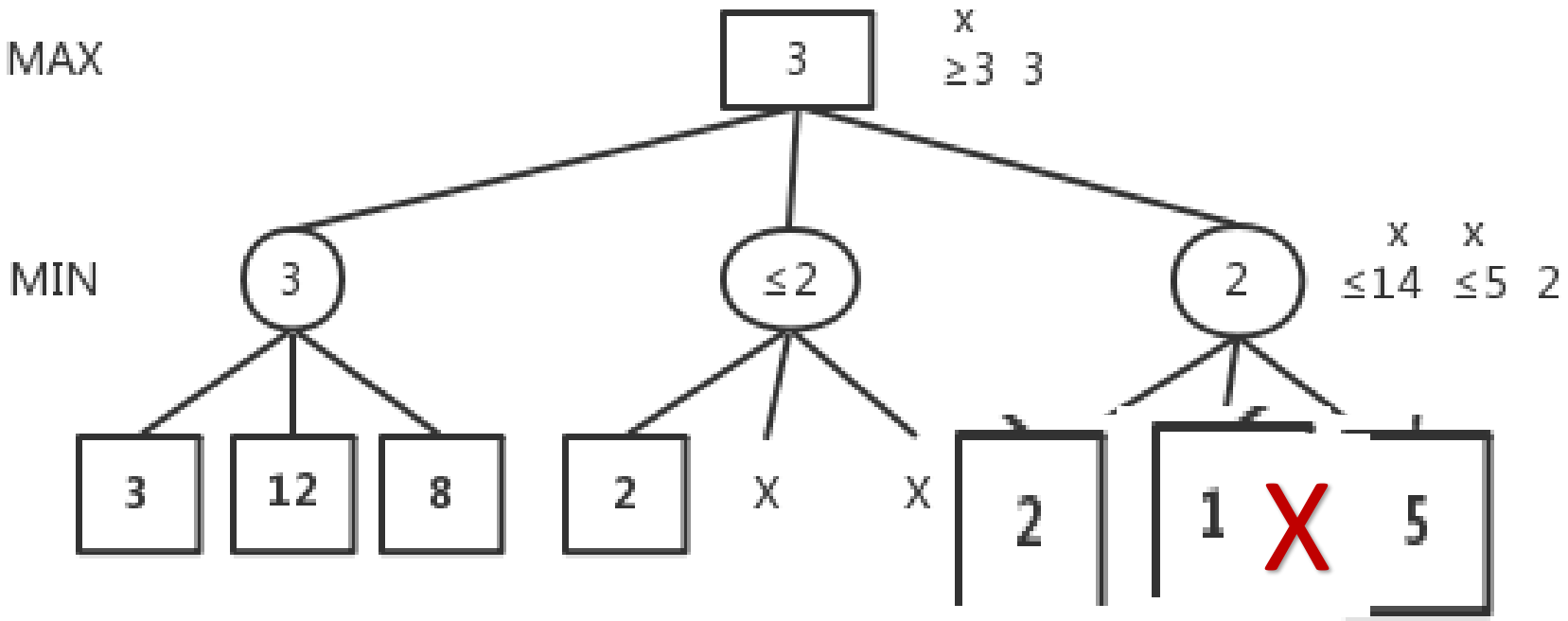
图中MIN选手所在的节点C下属分支4和6与根节点最终优化决策的取值无关，可不被访问。



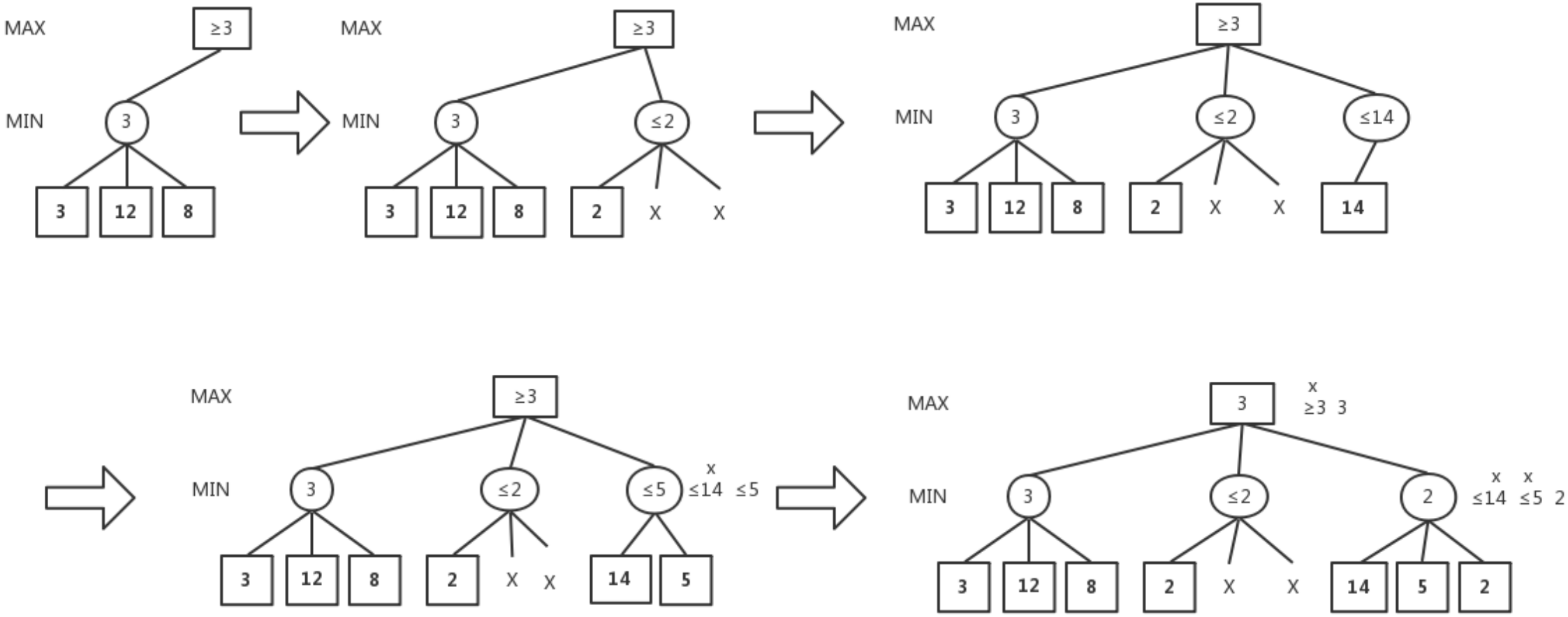
对抗搜索： Alpha-Beta 剪枝搜索



对抗搜索： Alpha-Beta 剪枝搜索



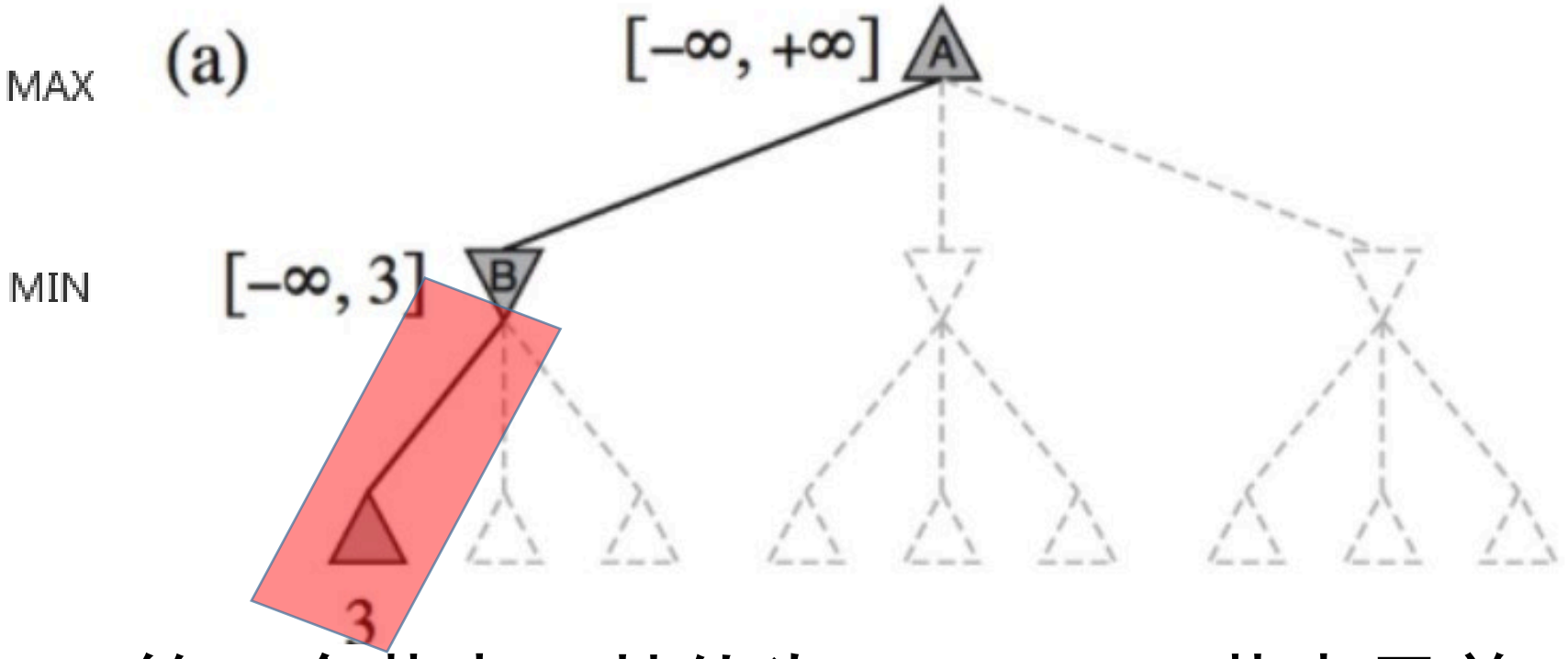
对抗搜索： Alpha-Beta 剪枝搜索



Alpha值(α)	MAX节点目前得到的最高收益
Beta值(β)	MIN节点目前可给对手的最小收益
α和β的值初始化分别设置为$-\infty$和∞	

对抗搜索：Alpha-Beta 剪枝搜索

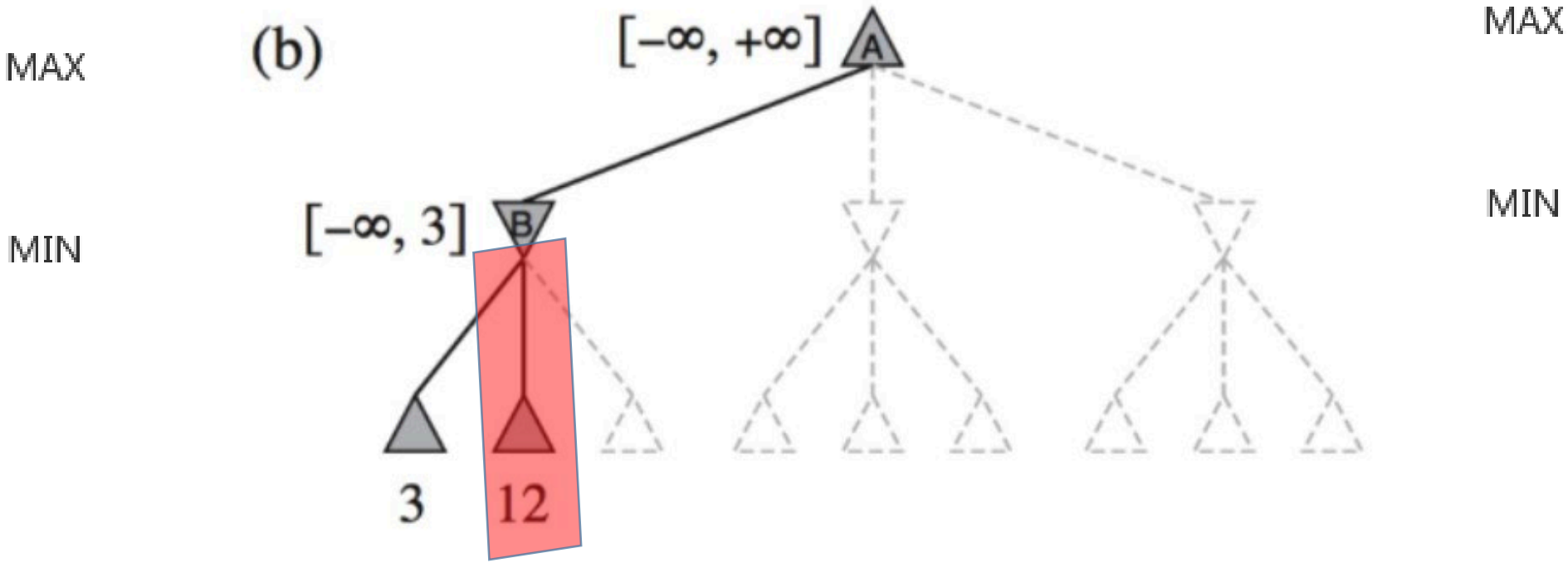
- 从α和β的变化来理解剪枝过程



- B第一个节点，其值为3，B(MIN)节点目前可给对手的最小收益，β 为3.

对抗搜索： Alpha-Beta 剪枝搜索

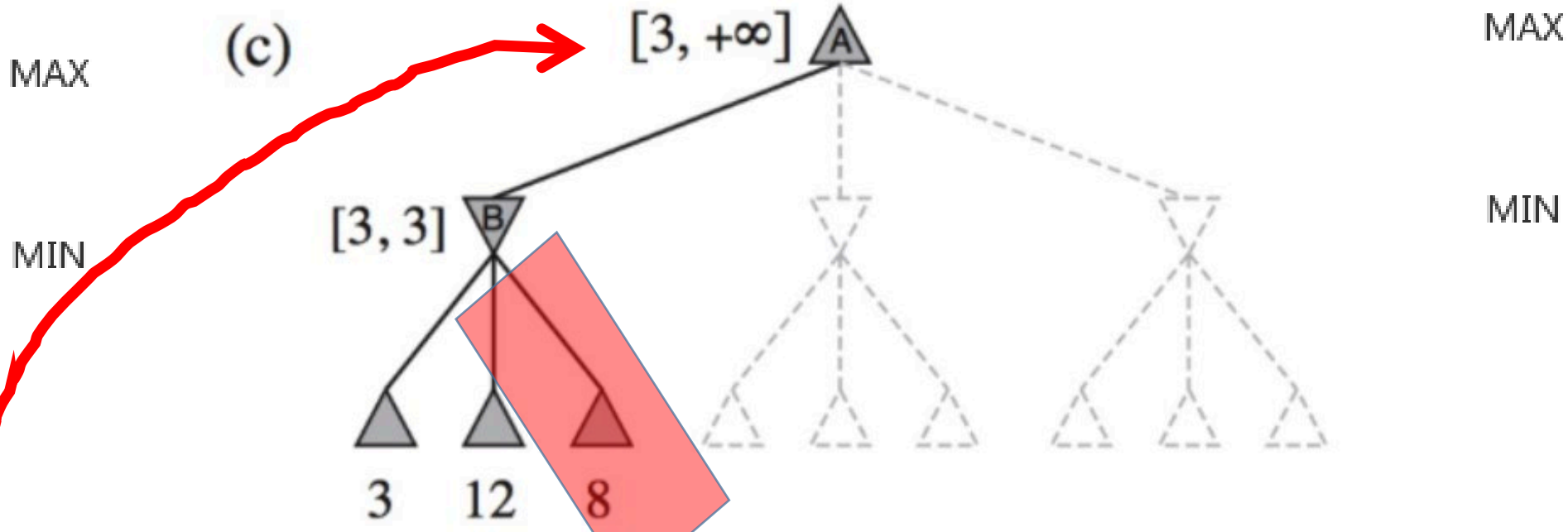
- 从α和β的变化来理解剪枝过程



- B第二个节点，其值为12(>3)，**B(MIN)**节点目前可给对手的最小收益，β 为3.

对抗搜索： Alpha-Beta 剪枝搜索

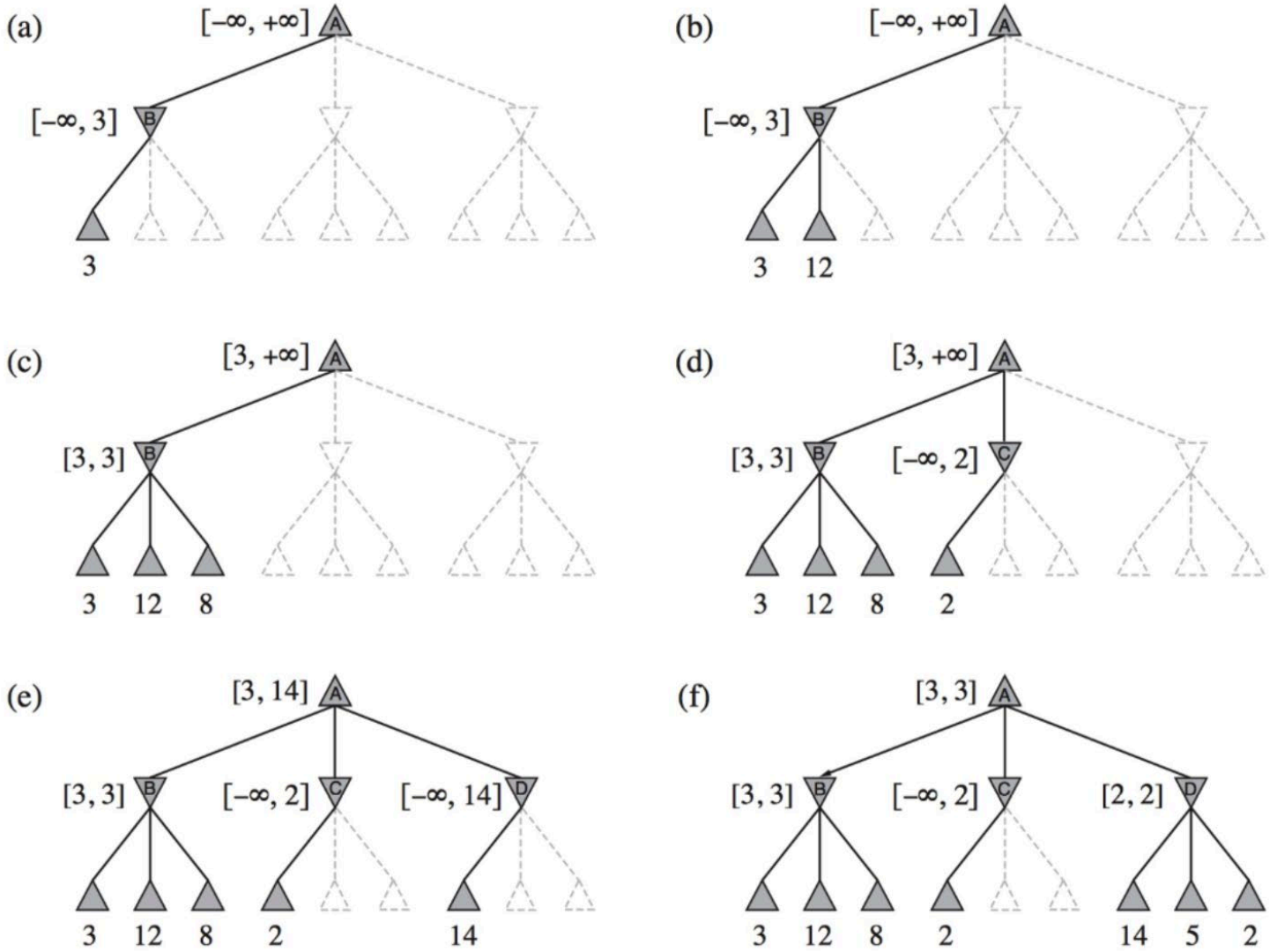
- 从α和β的变化来理解剪枝过程



- B第三个节点，其值为8(>3)，**B(MIN)**节点目前可给对手的最小收益，β 为3.
- A的第一个节点B，A节点目前得到的最高收益是3，α为3

对抗搜索： Alpha-Beta 剪枝搜索

- 从α和β的变化来理解剪枝过程



对抗搜索：如何利用Alpha-Beta 剪枝

Alpha值(α)	玩家MAX(根节点)目前得到的最高收益
	假设n是MIN节点，如果n的一个后续节点可提供的收益小于α，则n及其后续节点可被剪枝

对抗搜索：如何利用Alpha-Beta 剪枝

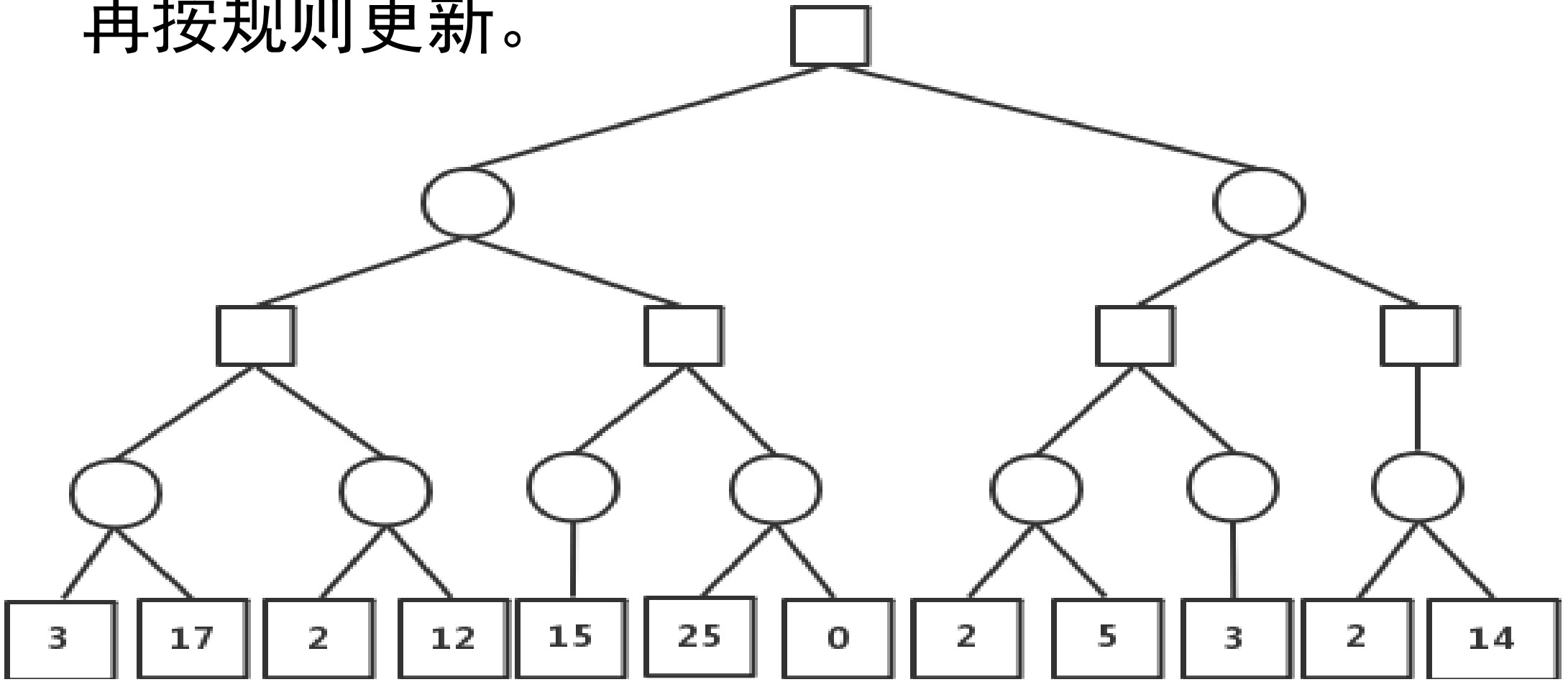
- α为可能解法的最小下界
- β为可能解法的最大上界

对抗搜索：如何利用Alpha-Beta 剪枝

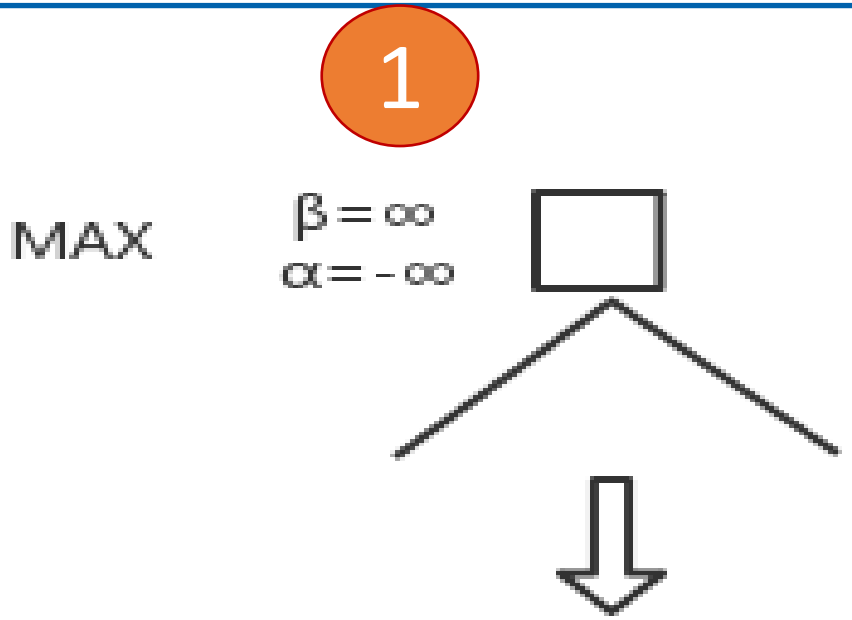
- α为可能解法的最小下界
- β为可能解法的最大上界

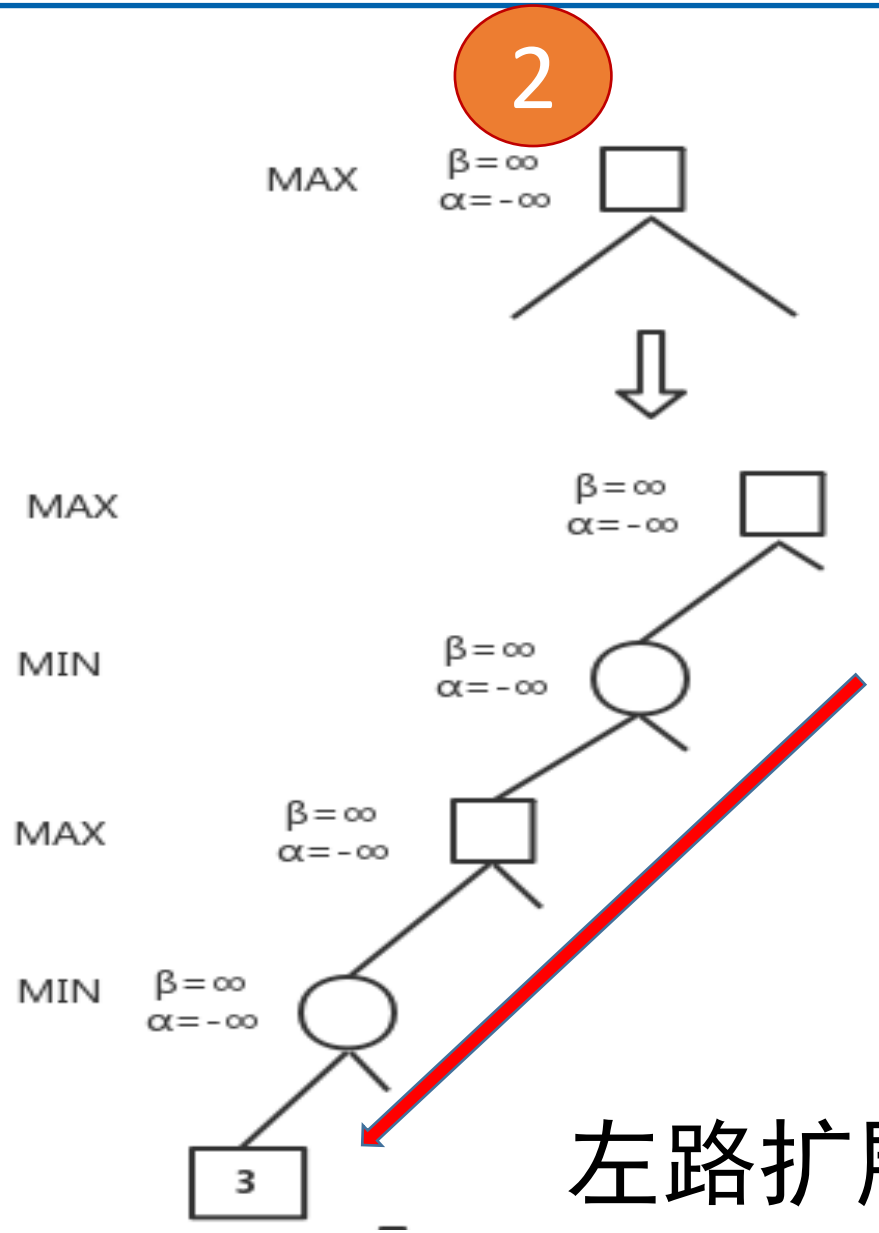
对抗搜索：Alpha-Beta 剪枝搜索示意

没有必要枚举，可以**先继承父节点**的α, β，
再按规则更新。



对抗搜索： Alpha-Beta 剪枝搜索示意

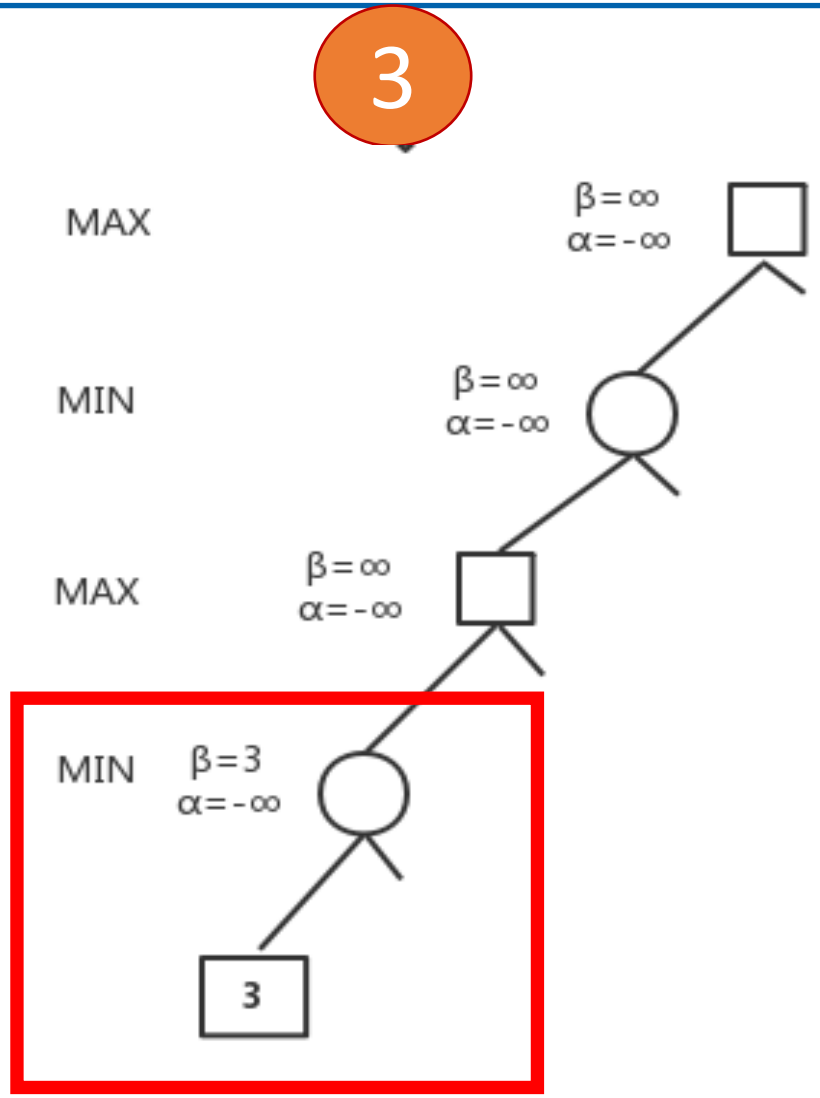


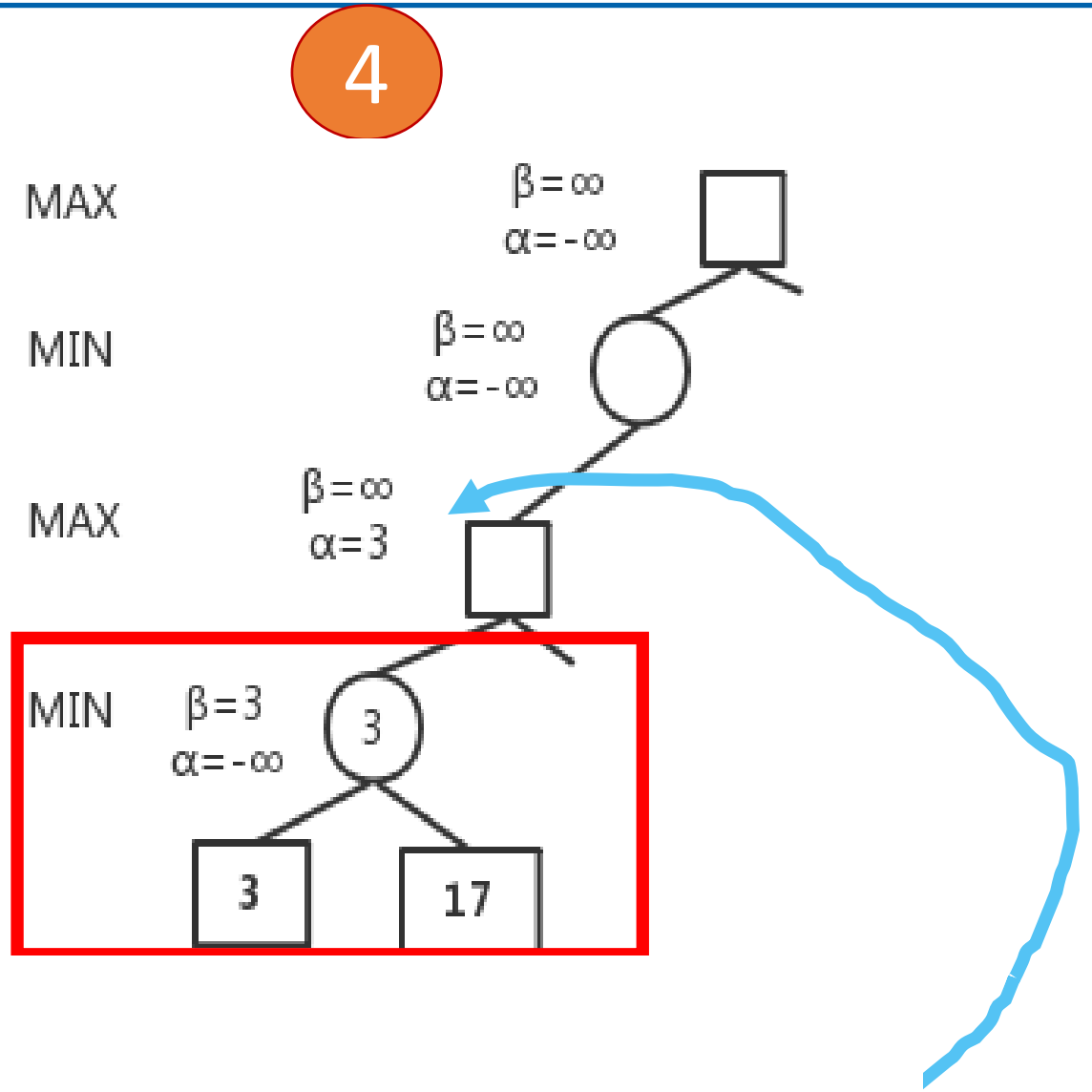


α, β赋值

左路扩展到底

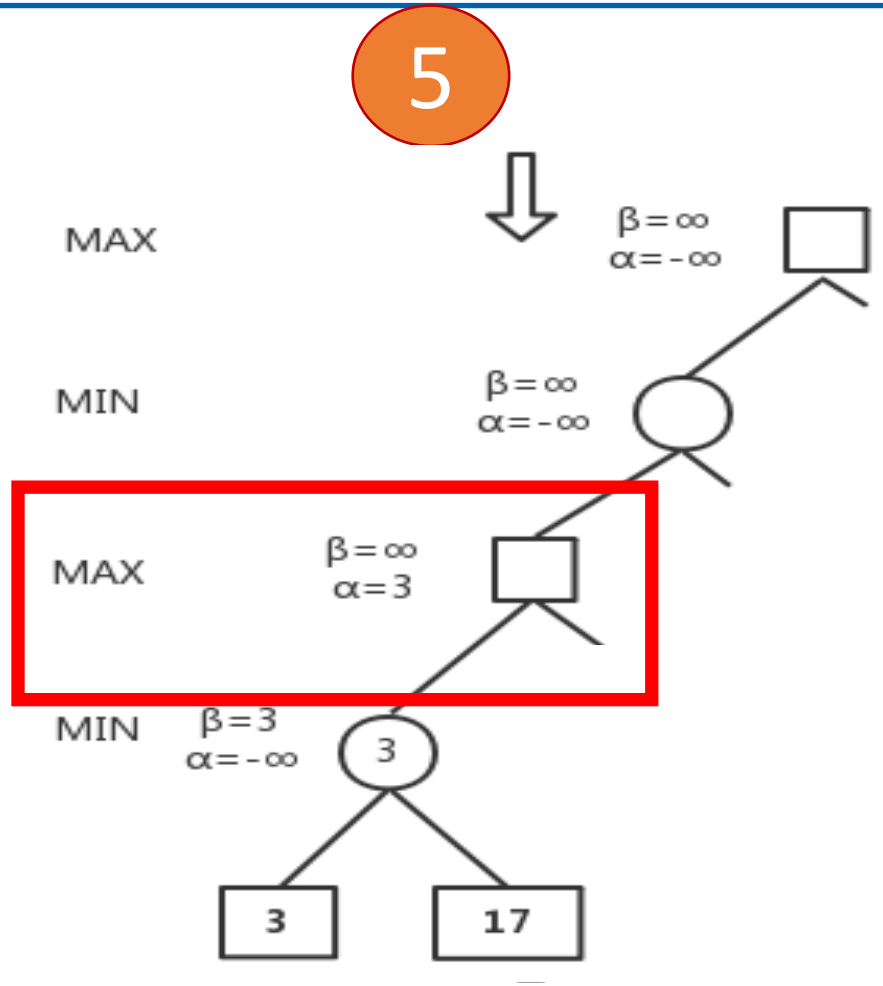
对抗搜索： Alpha-Beta 剪枝搜索示意

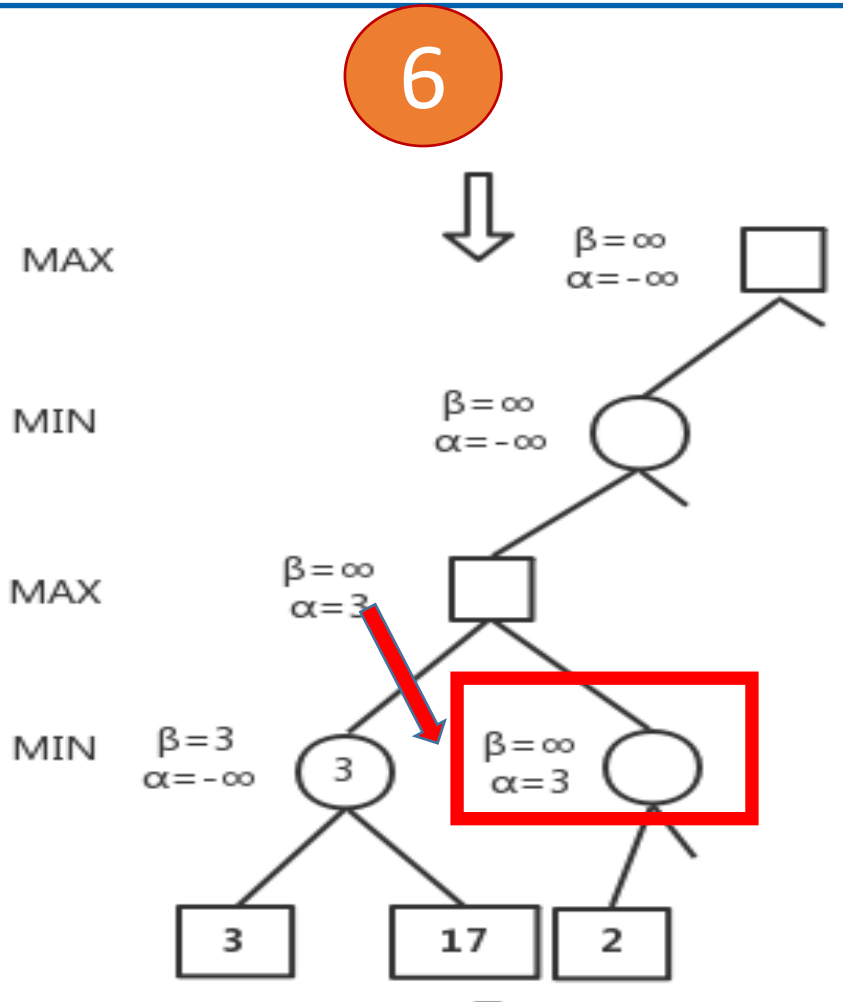




MIN节点，$\beta = 3$

对抗搜索：Alpha-Beta 剪枝搜索示意

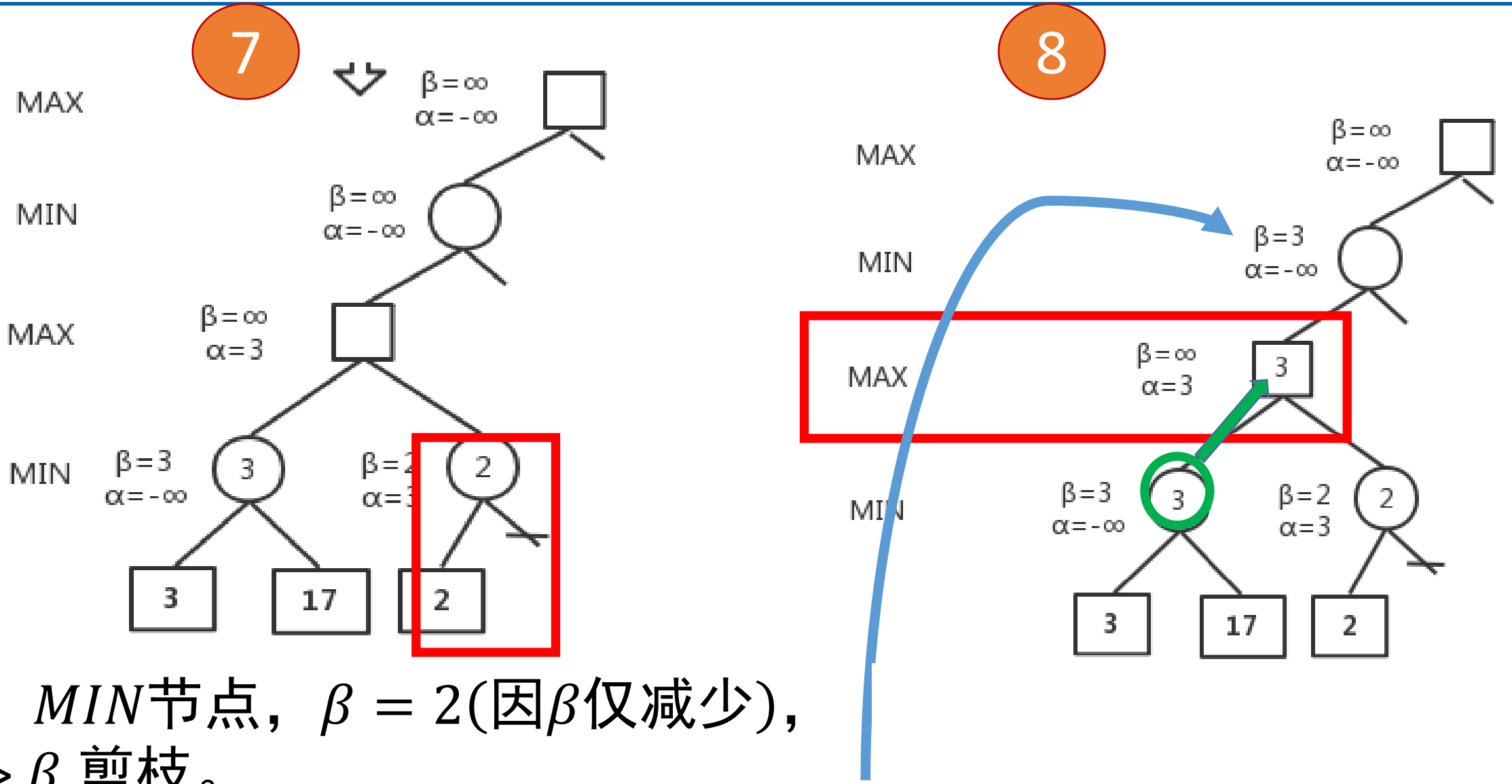




MAX节点，$\alpha = 3$

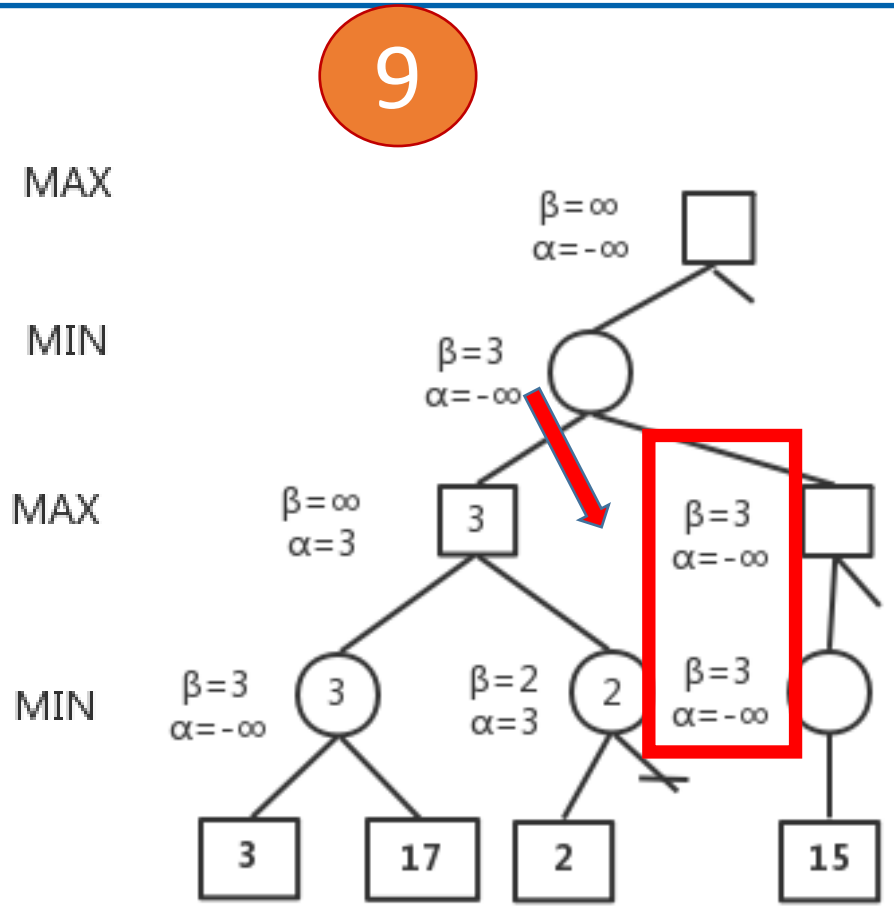
MIN节点，先继承，α为3

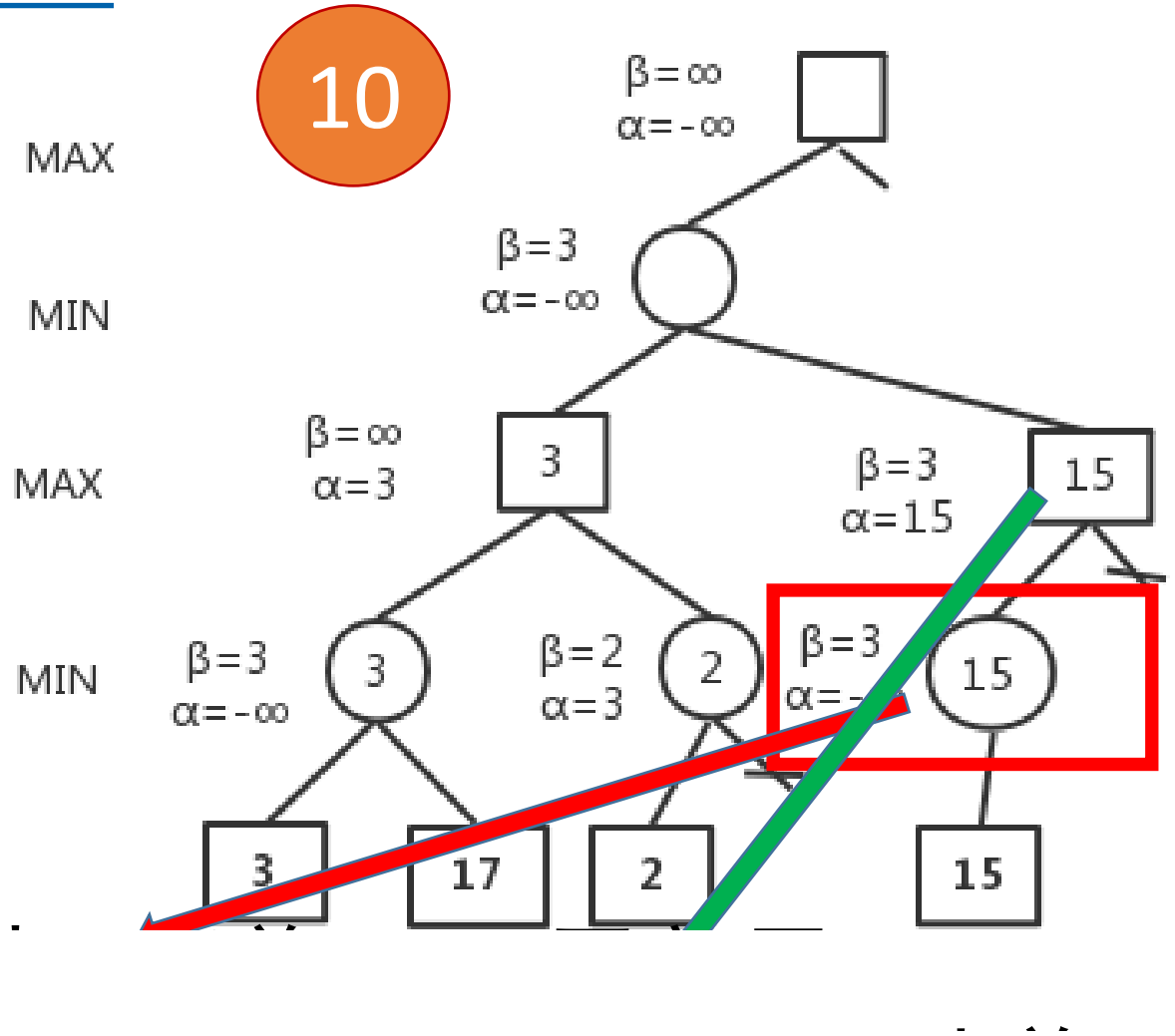
对抗搜索：Alpha-Beta 剪枝搜索示意



MIN节点，$\beta = 2$(因β仅减少)，$\alpha > \beta$,剪枝。

对抗搜索：Alpha-Beta 剪枝搜索示意

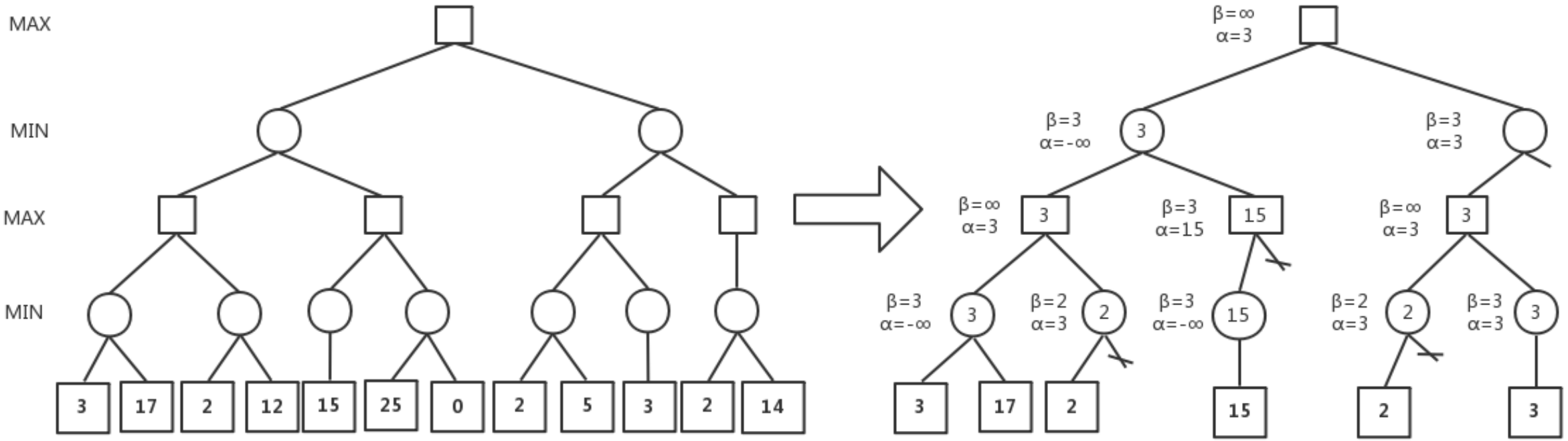




继承父节点，

对抗搜索：Alpha-Beta 剪枝搜索示意

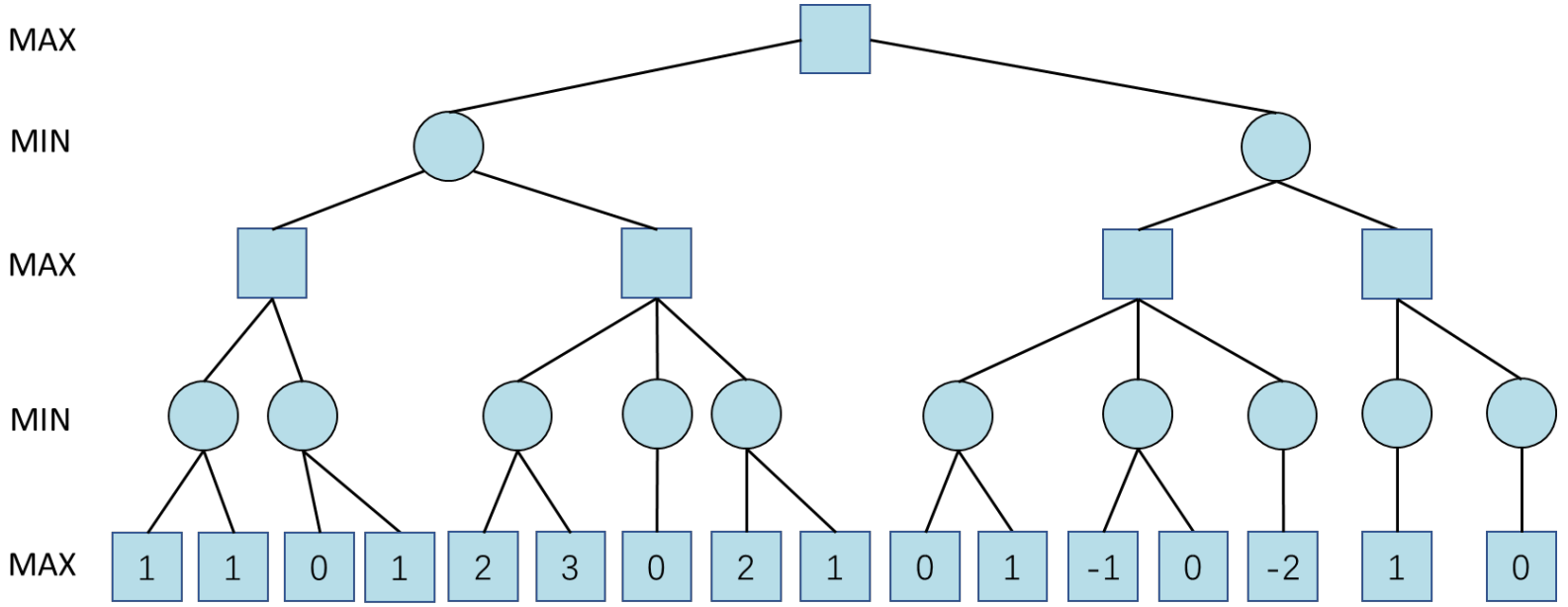
最终剪枝：



课上习题

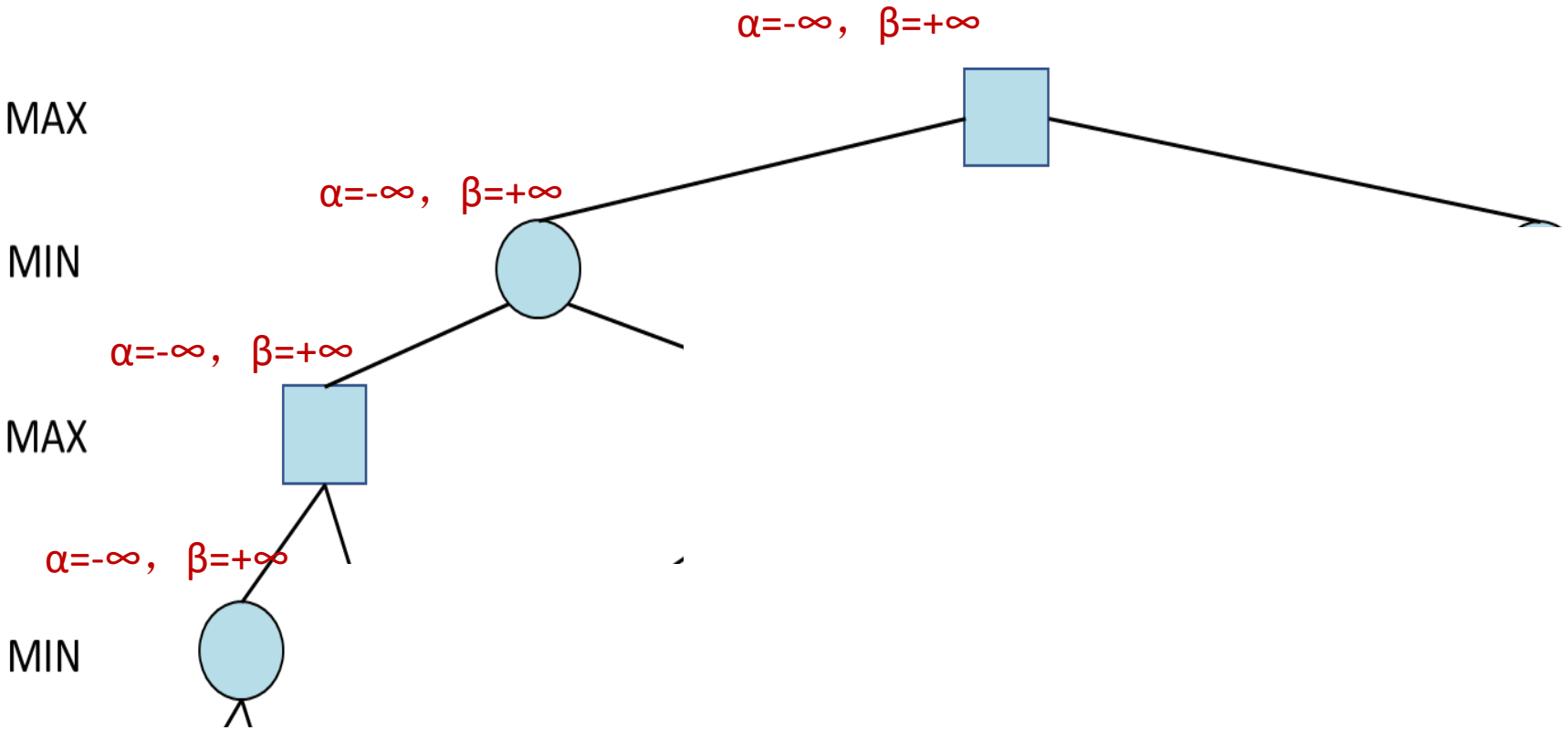
图2展示了一棵Minimax搜索树，可采用alpha-beta剪枝算法进行对抗搜索。假设对于每个节点的后继节点，算法按照从左向右的方向扩展。同时假设当alpha值等于beta值时，算法不进行剪枝。请问：

- (1)对图2(a)中所示搜索树进行搜索，请画出在算法结束时搜索树的状态，用“×”符号标出被剪枝的子树，并计算该算法扩展的节点数量。

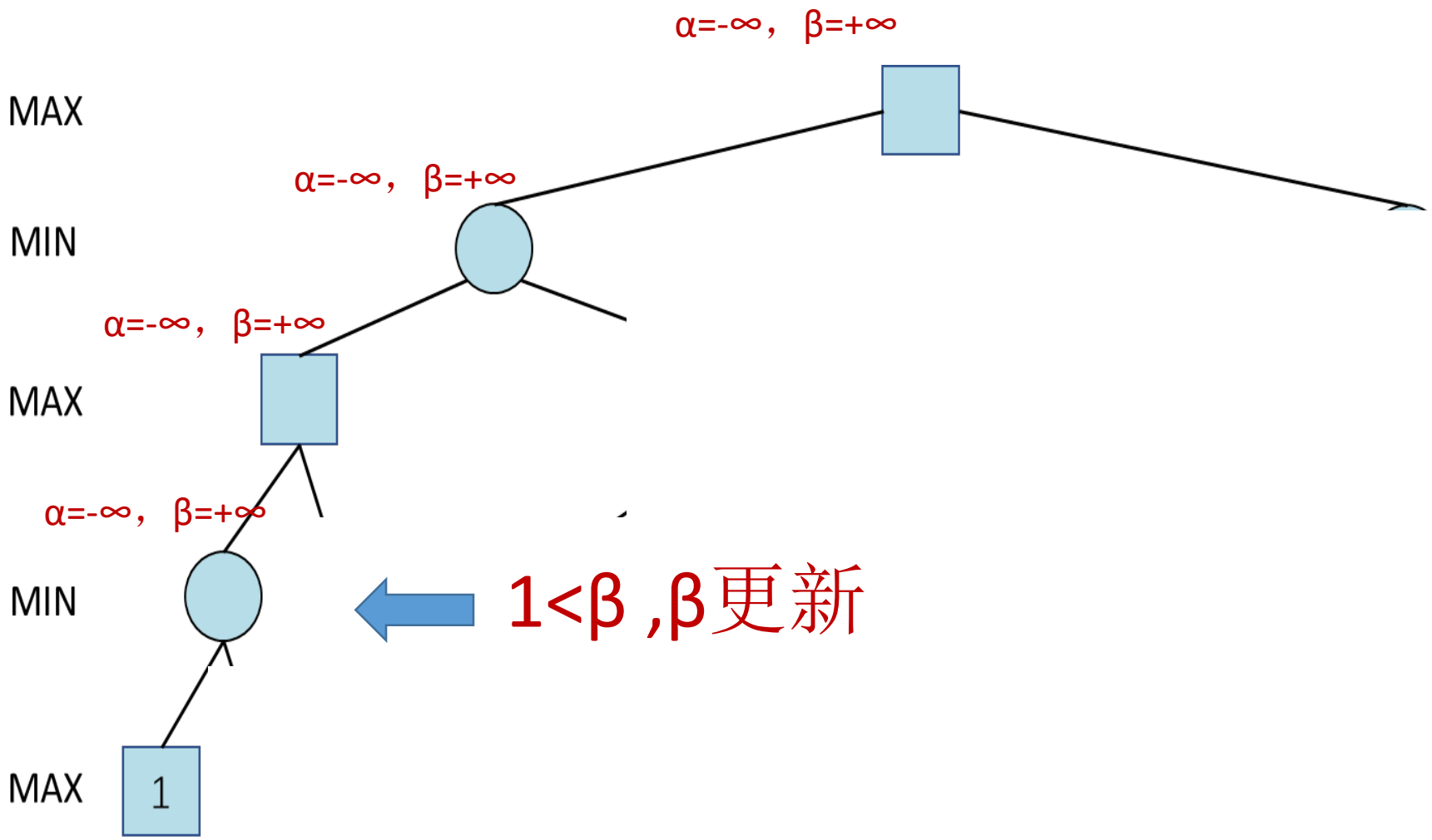


(a)

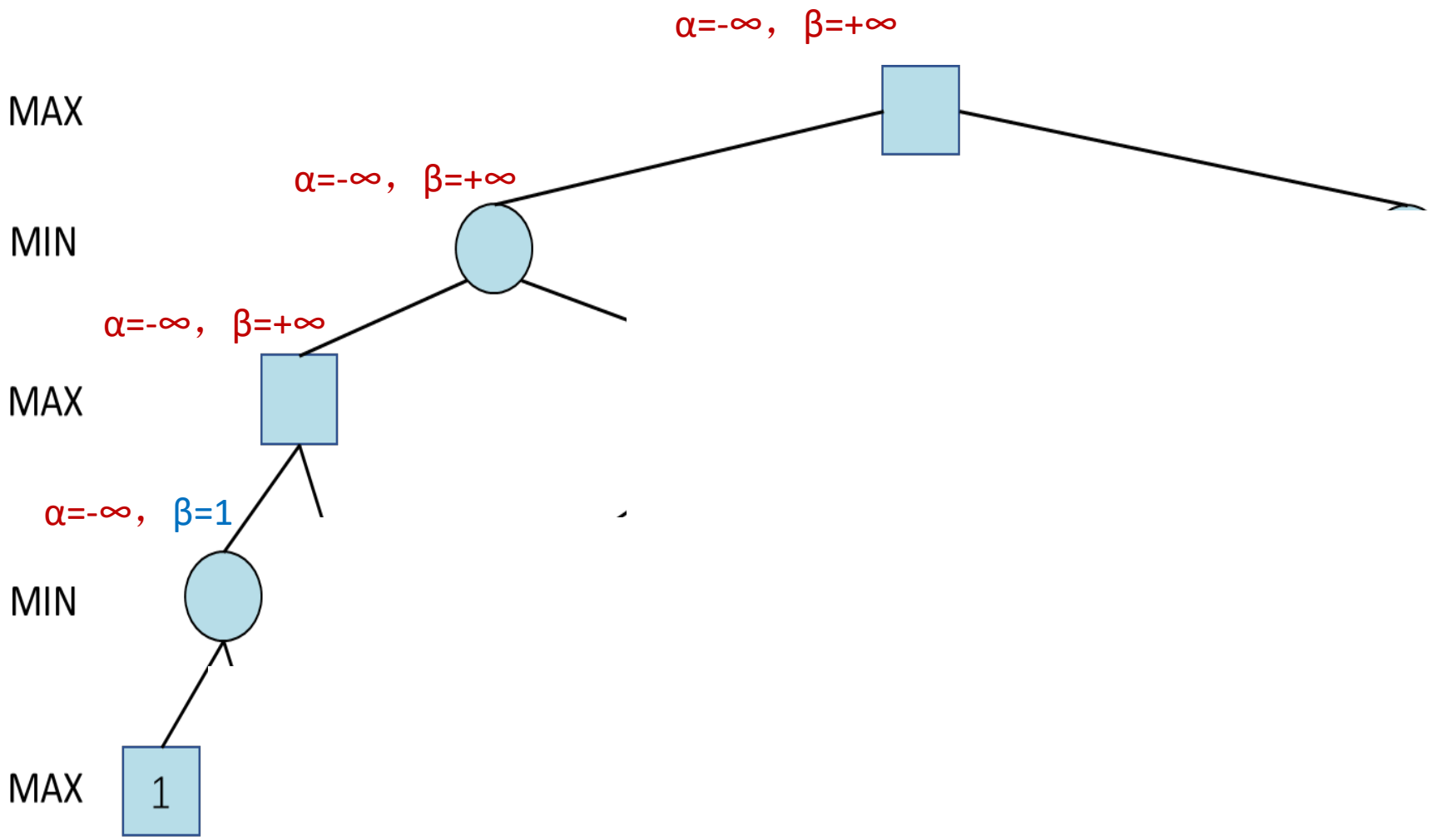
课上习题



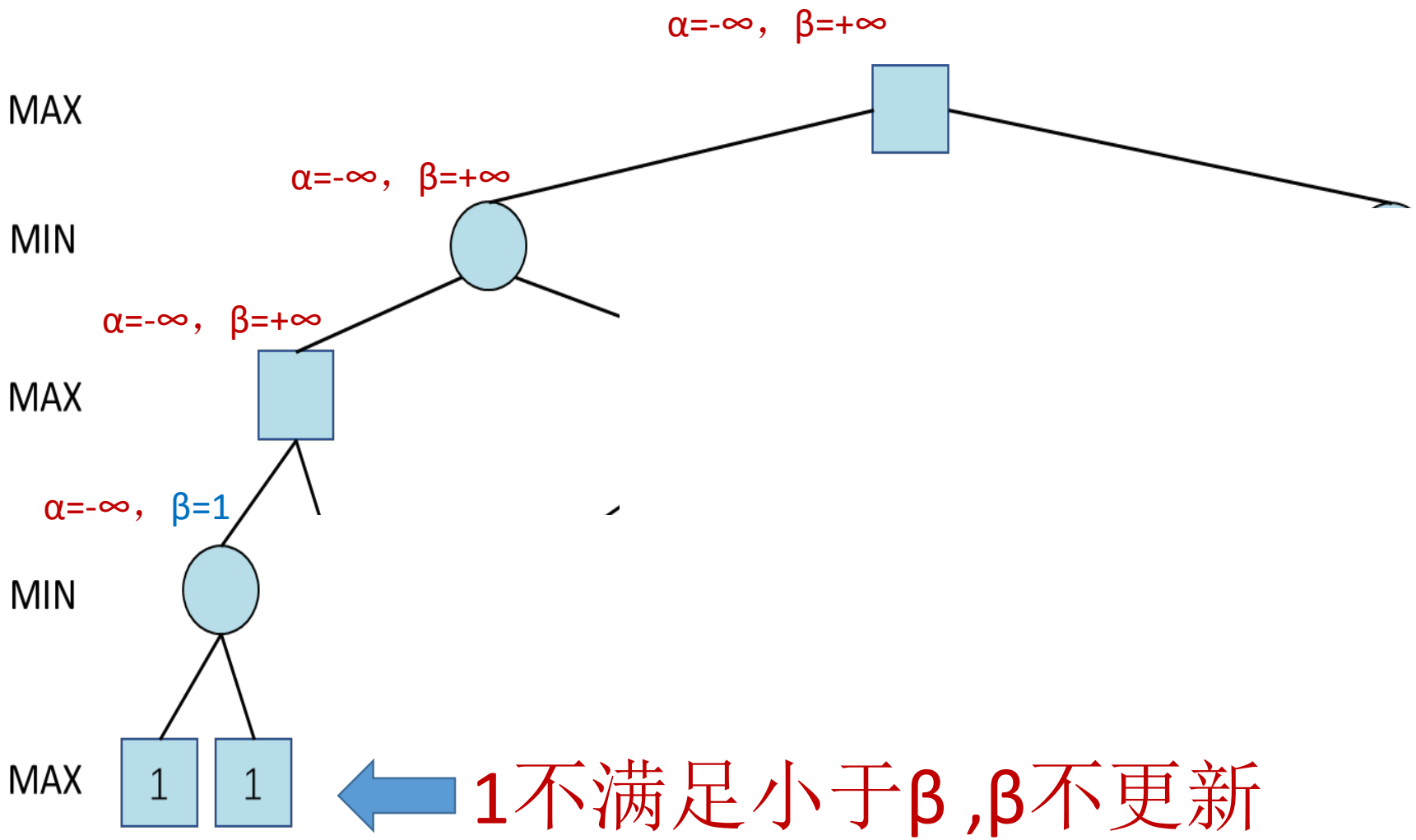
课上习题



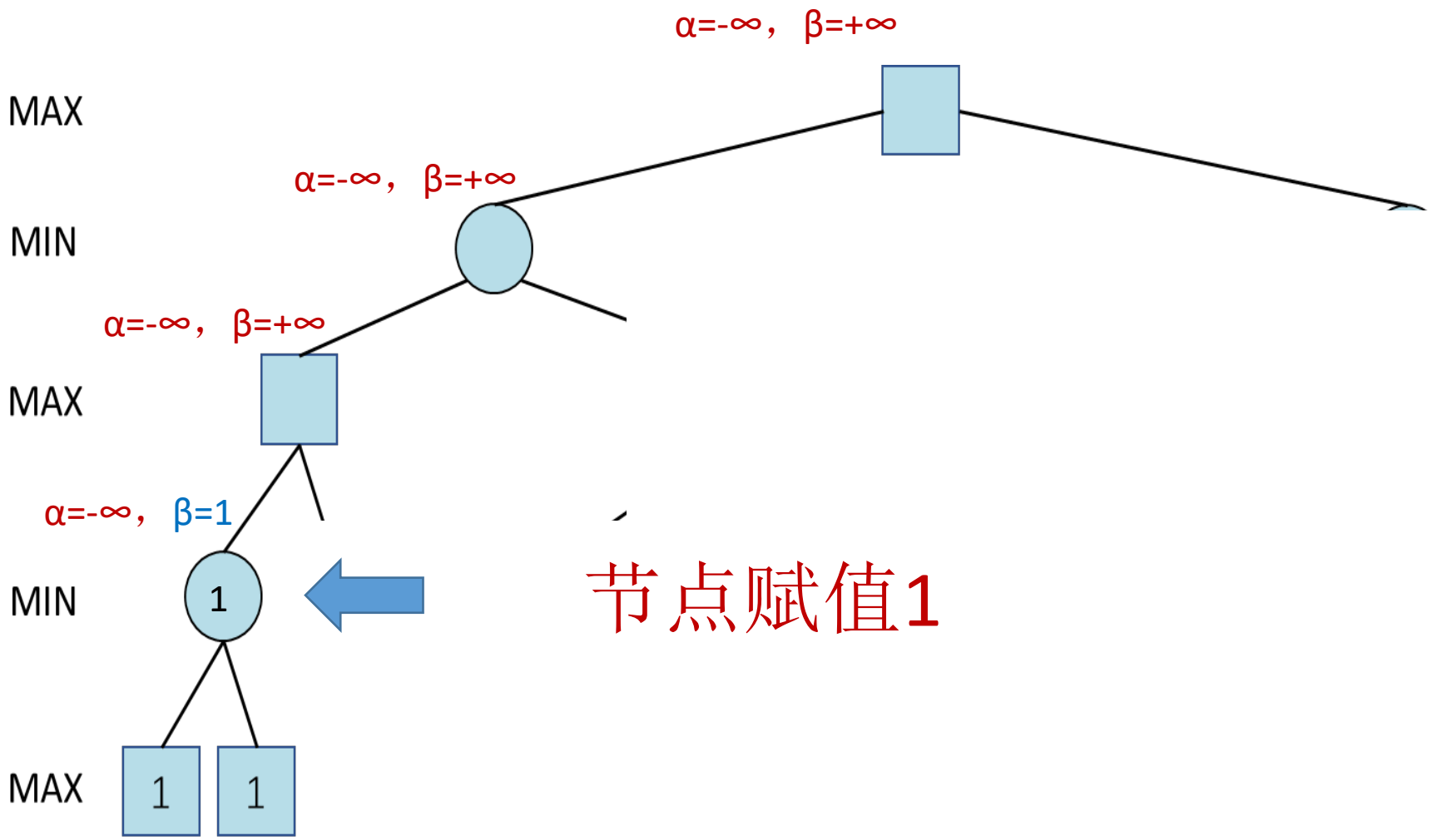
课上习题



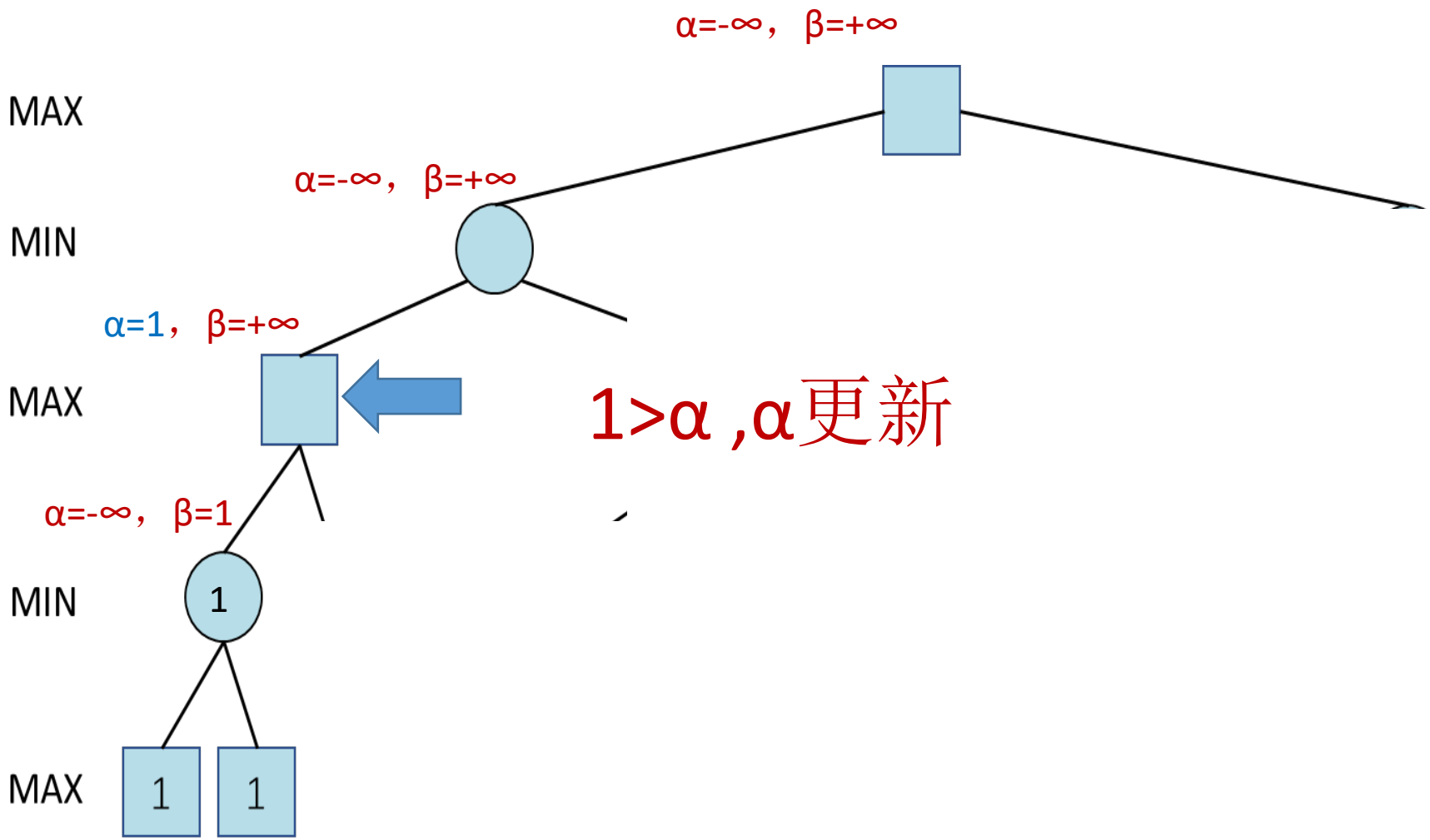
课上习题



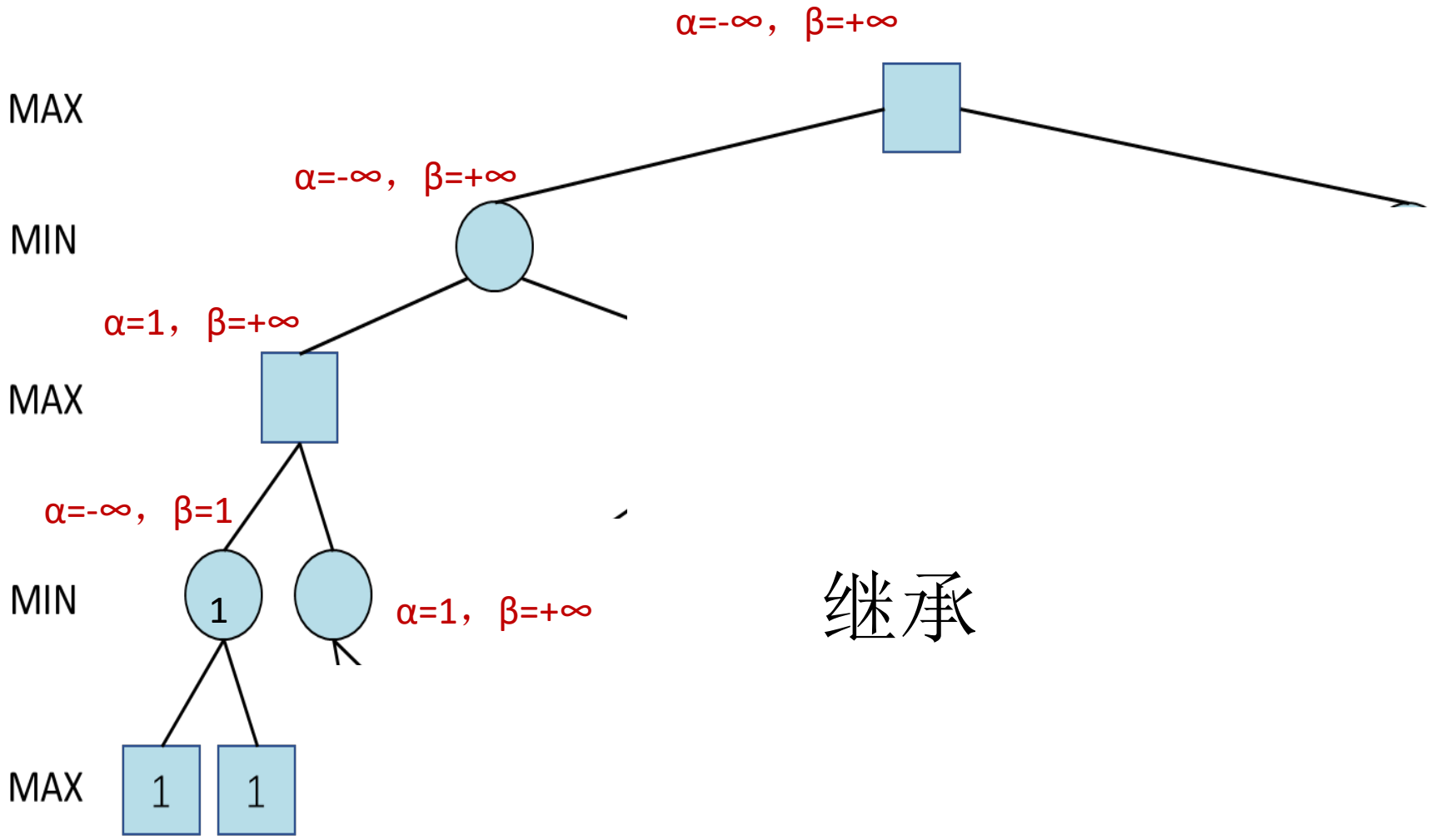
课上习题



课上习题

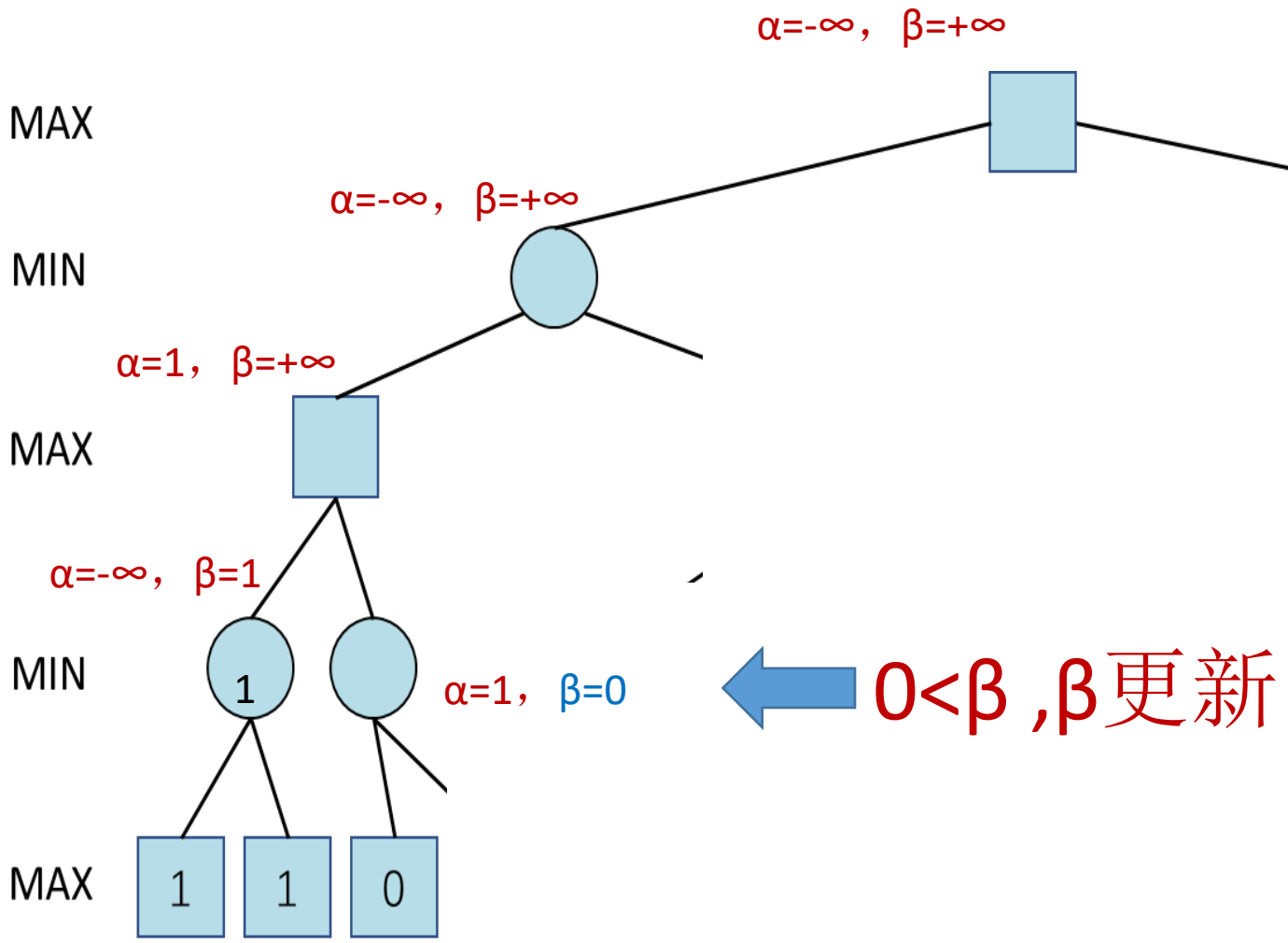


课上习题



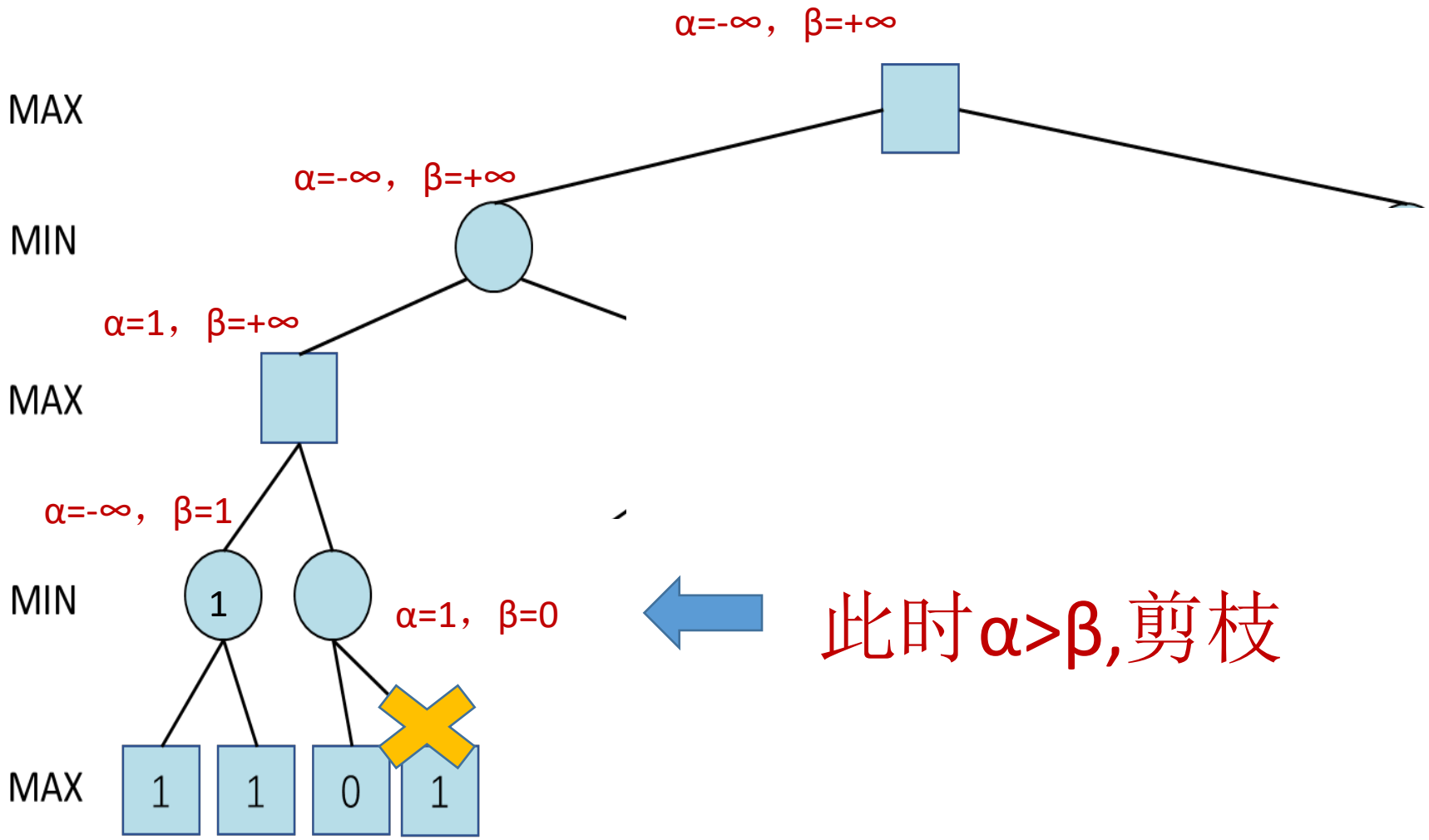
继承

课上习题

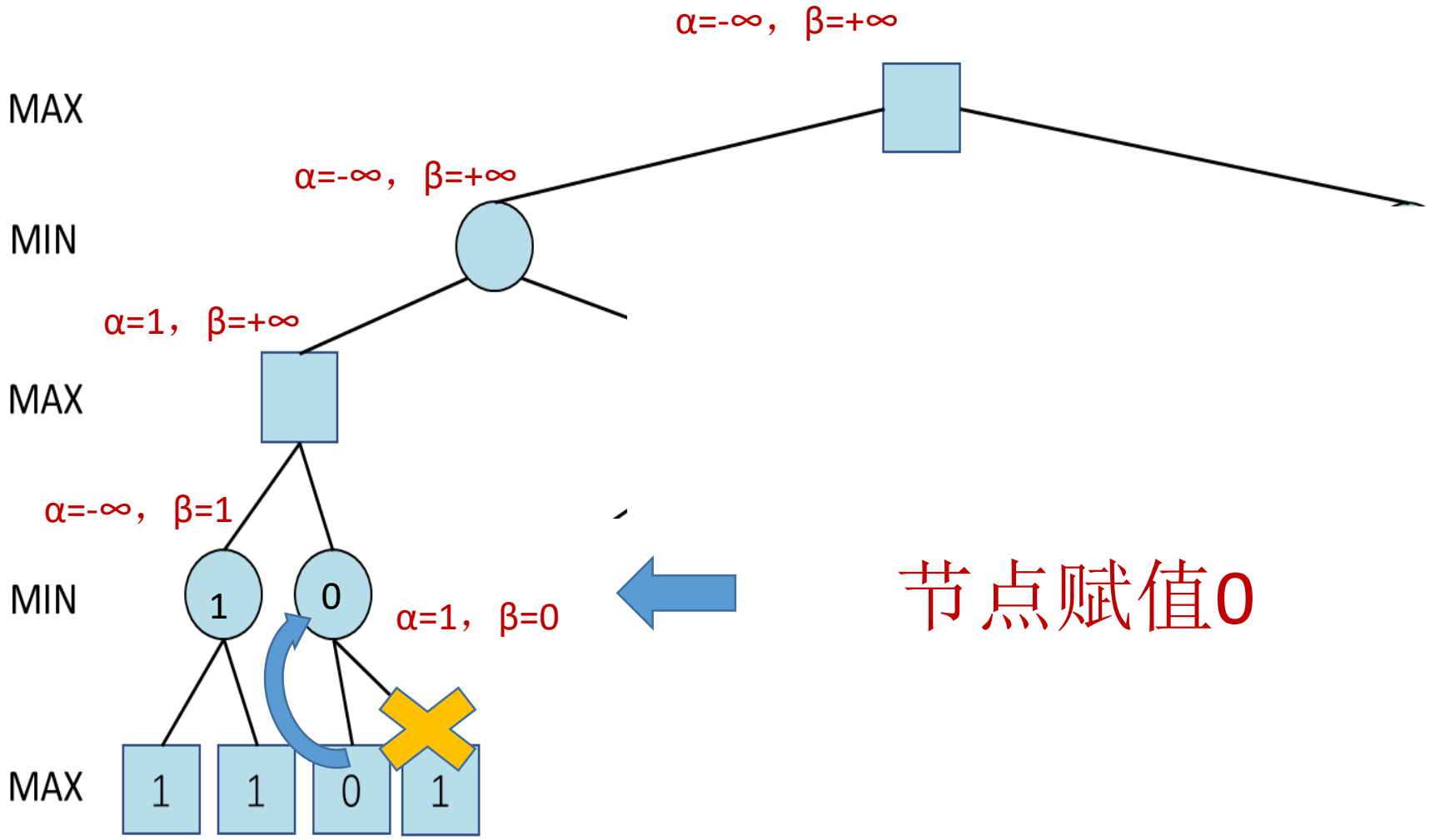


0<β ,β更新

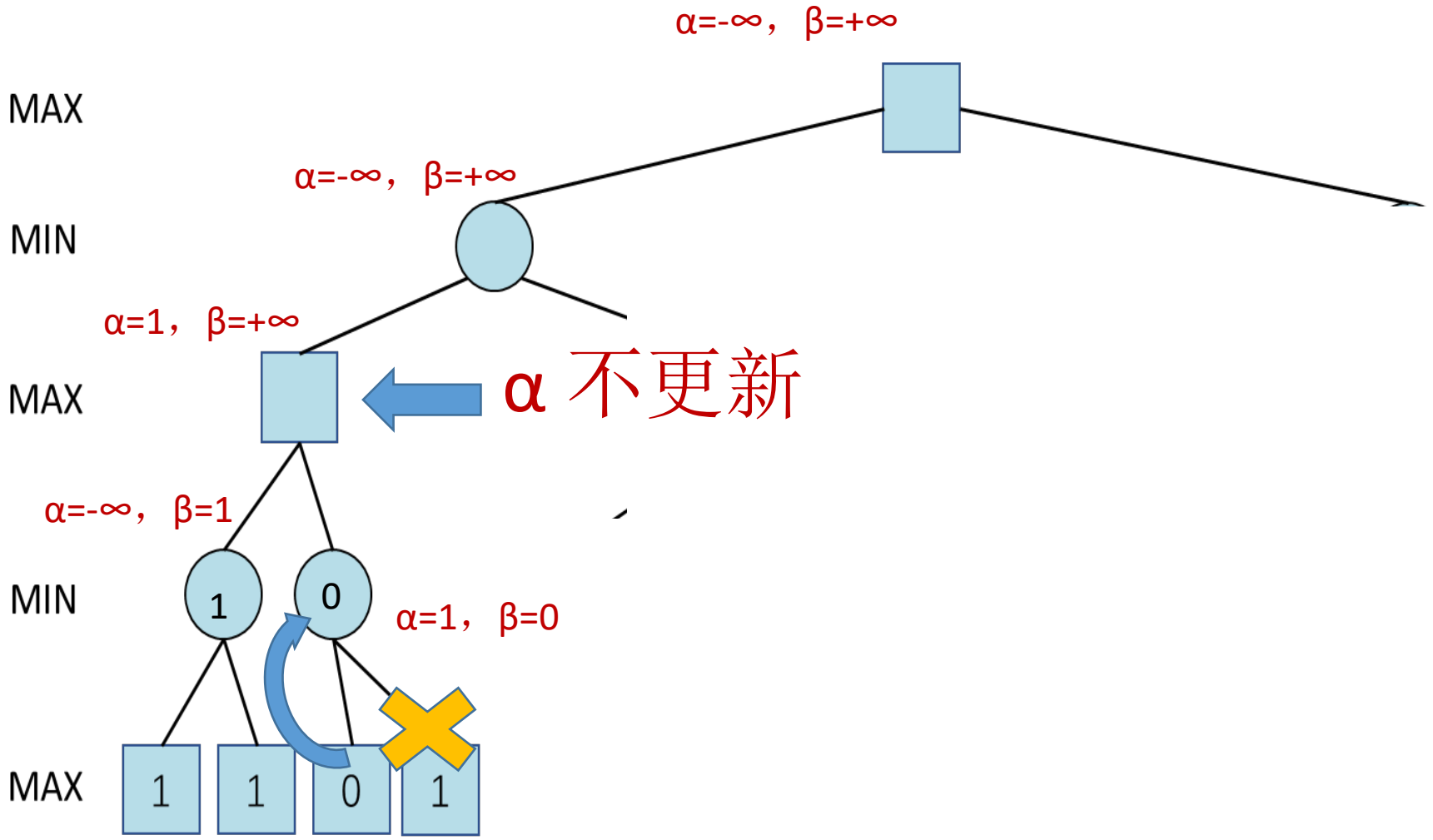
课上习题



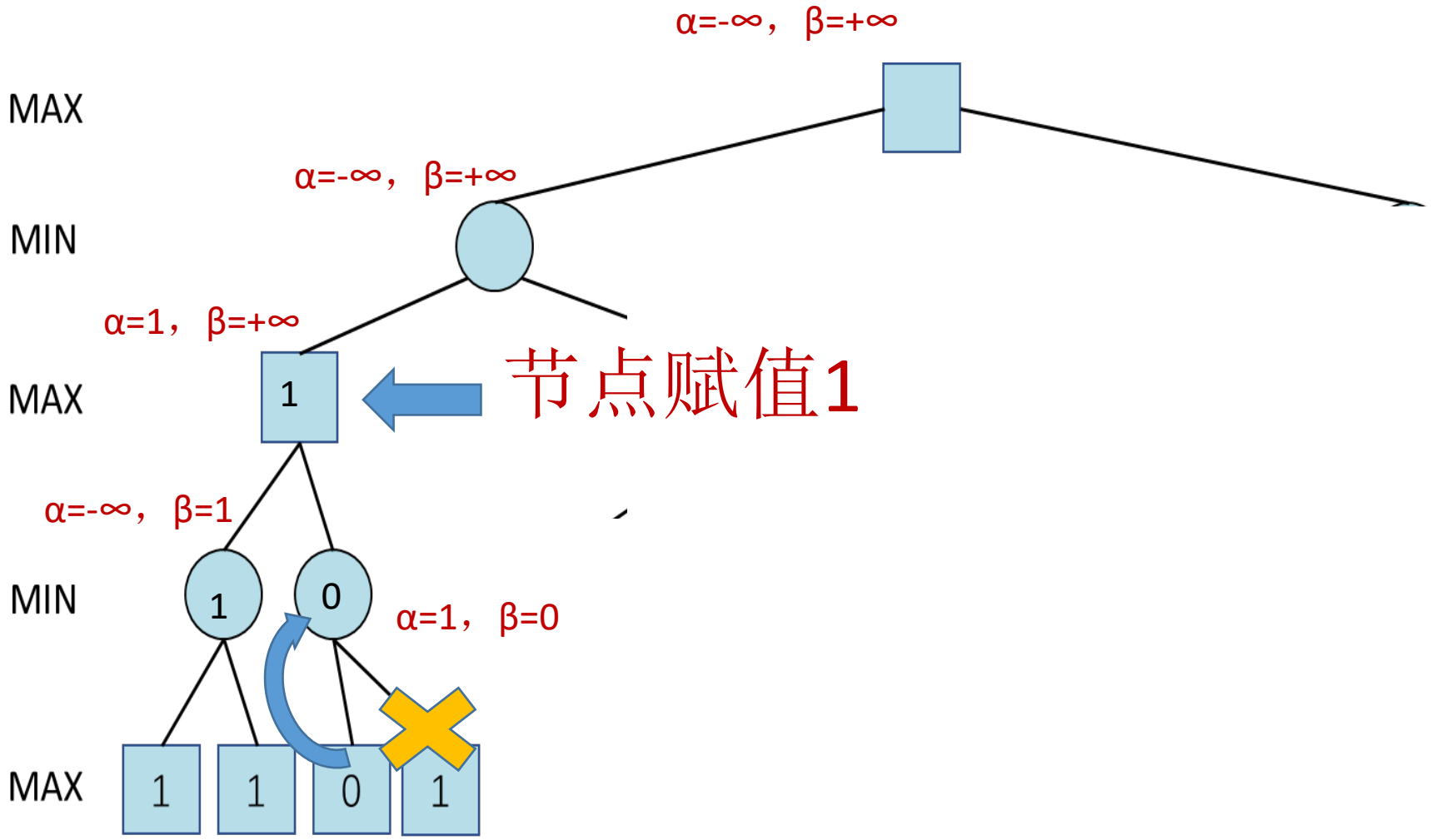
课上习题



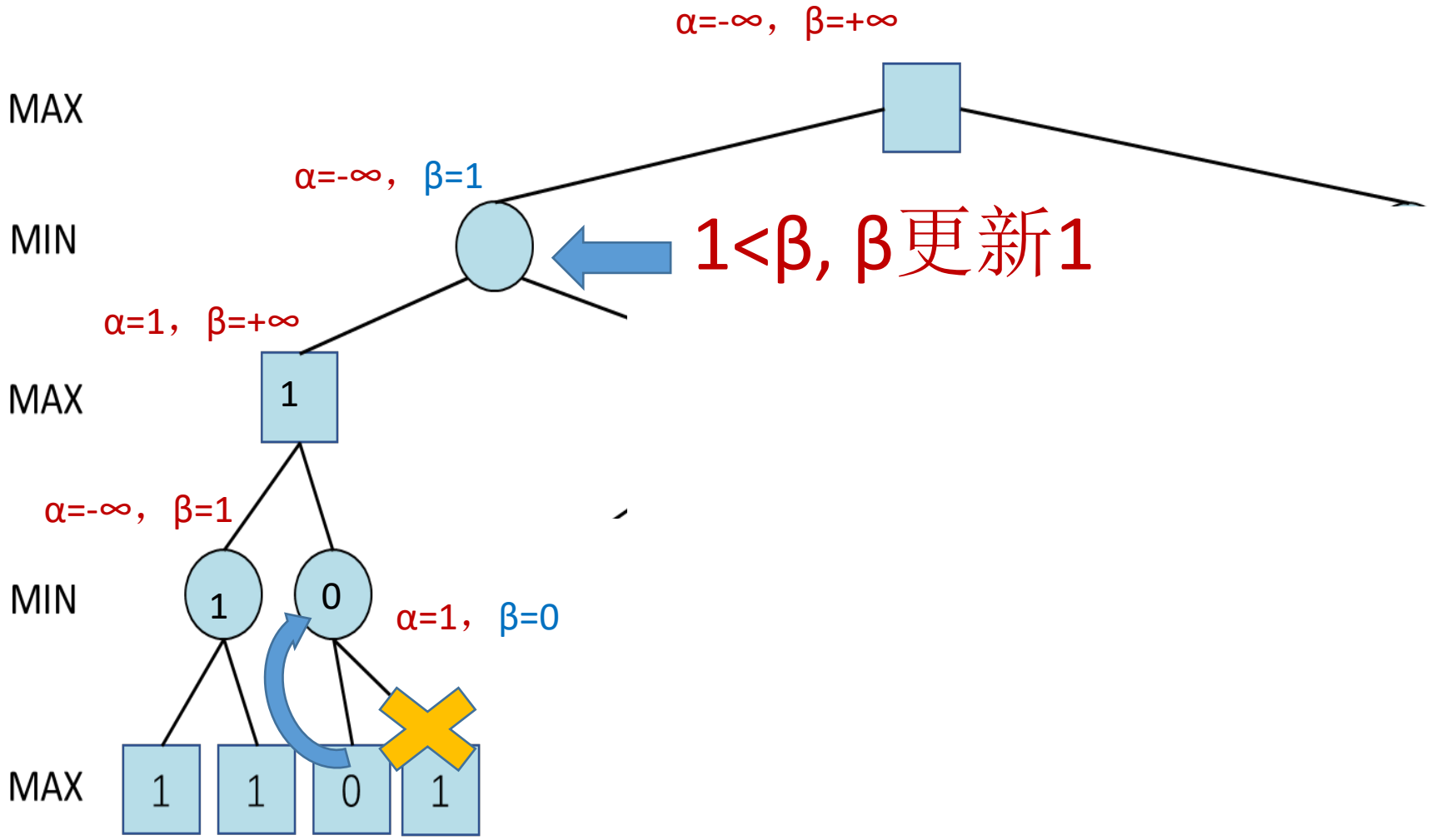
课上习题



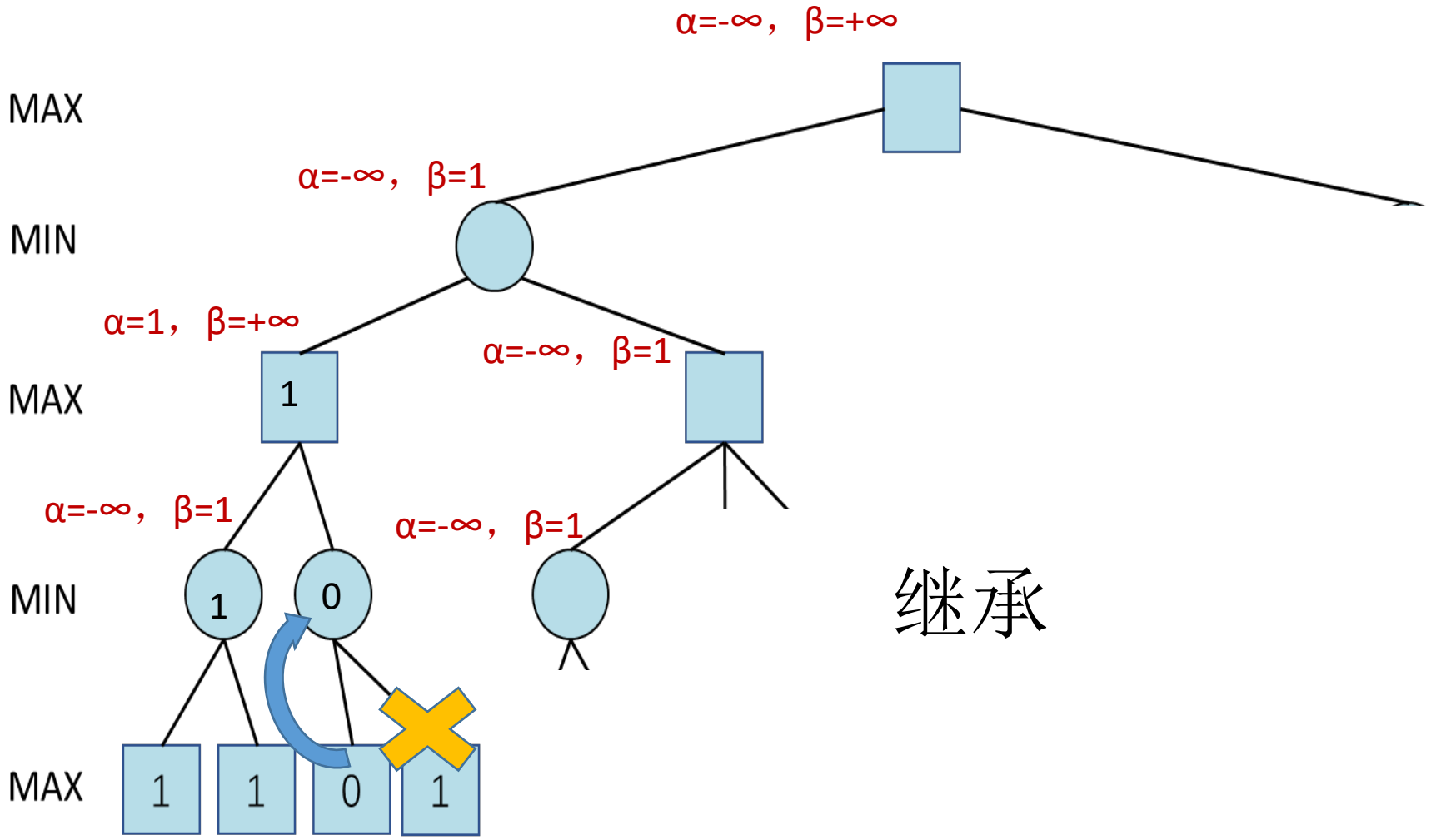
课上习题



课上习题

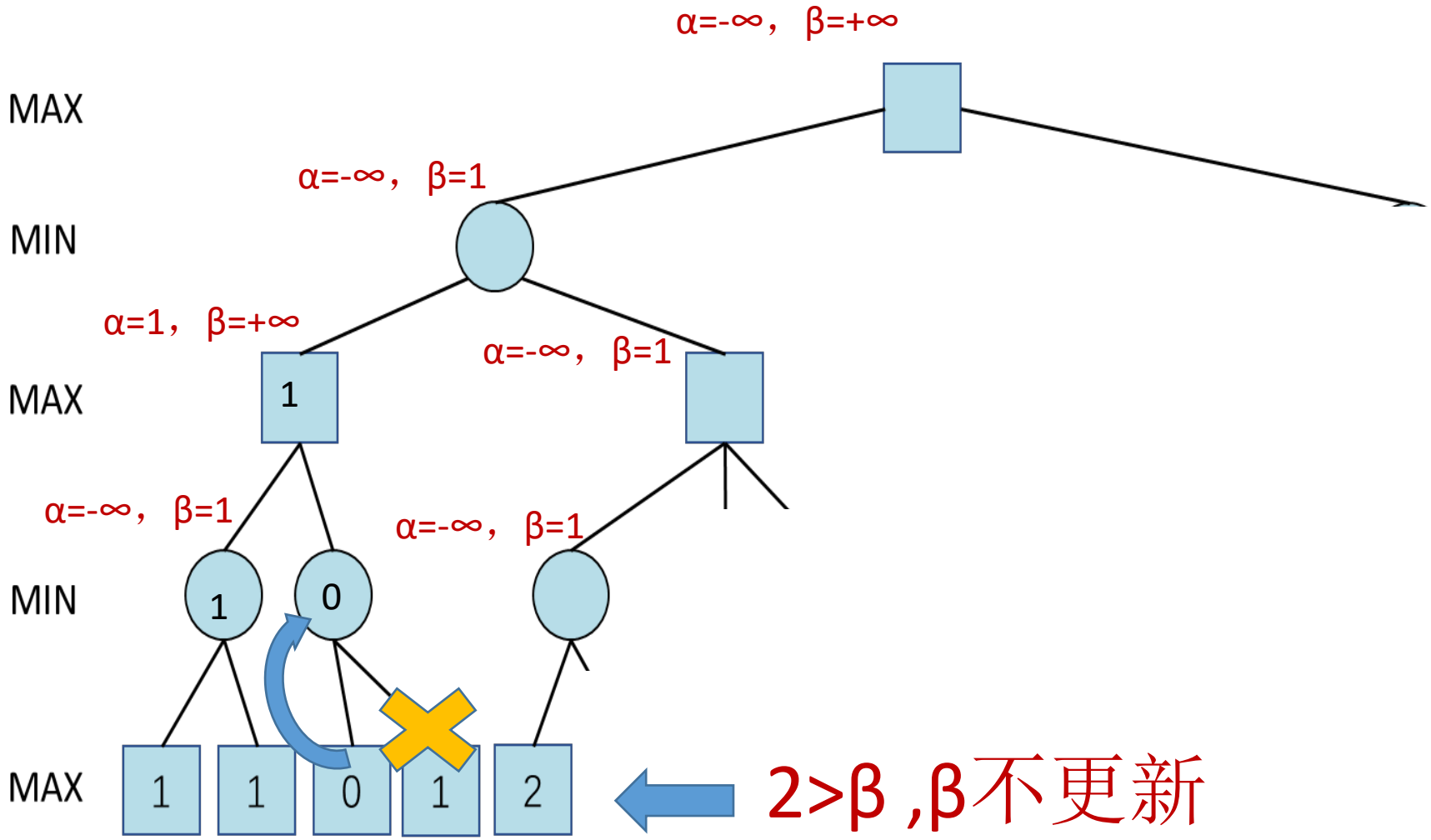


课上习题

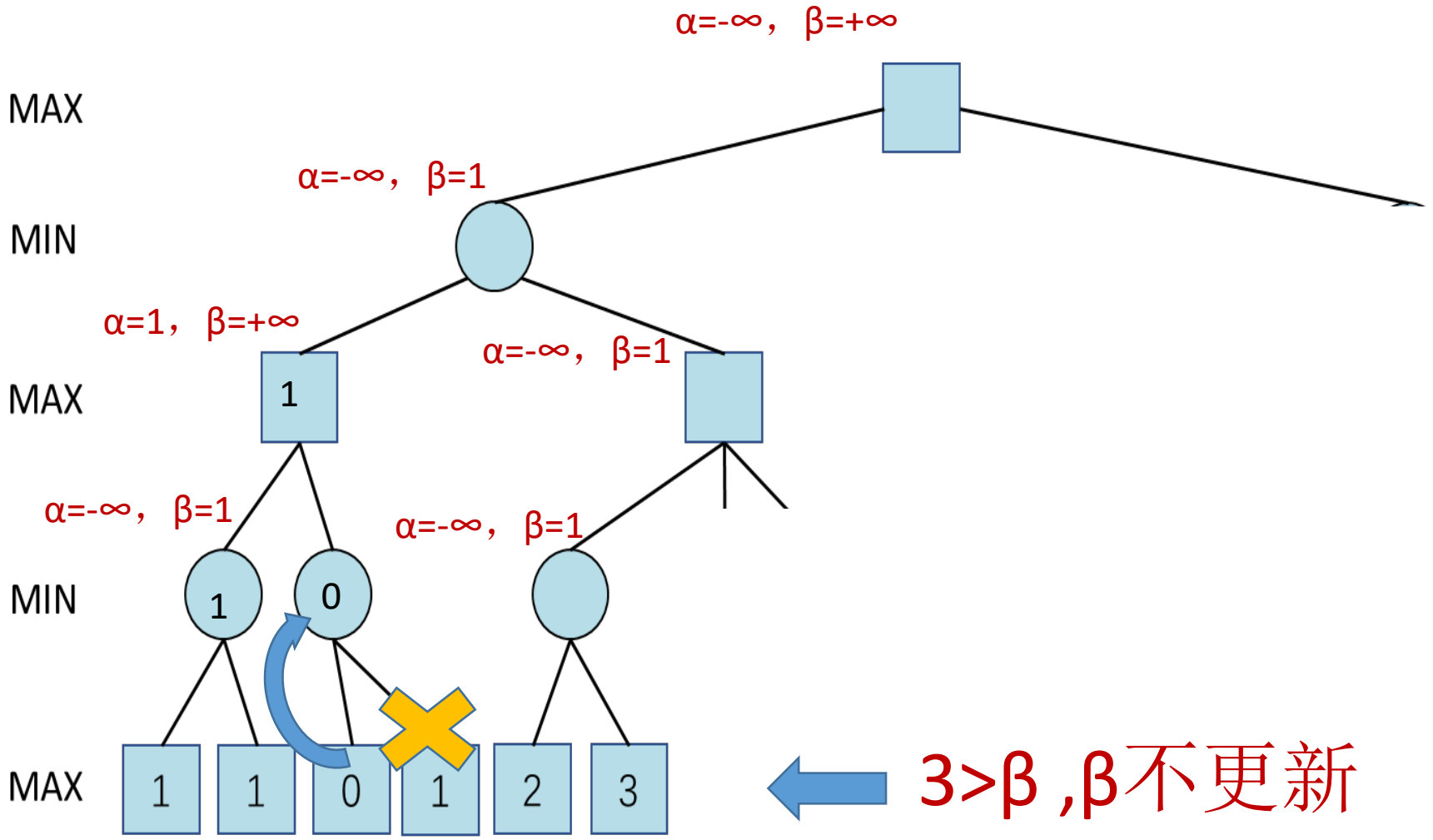


继承

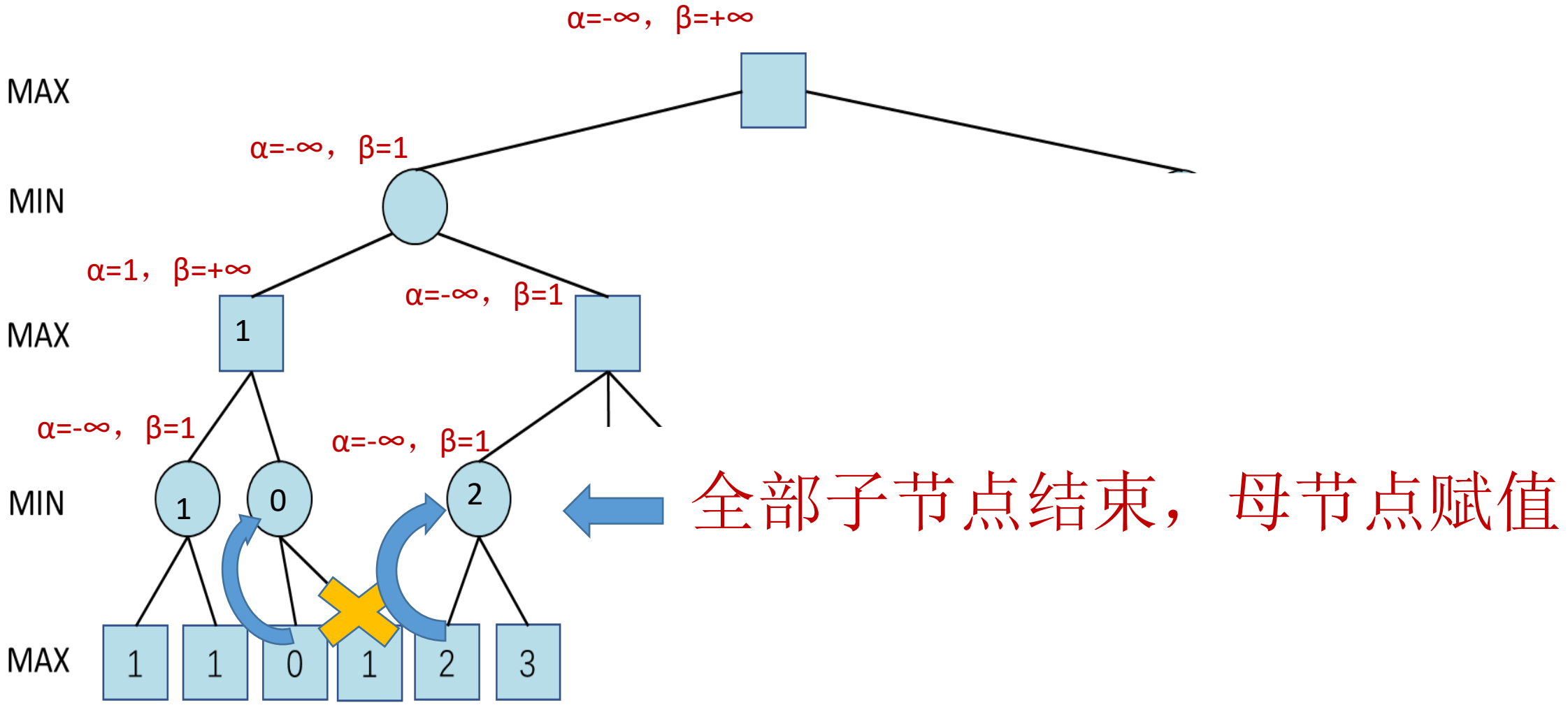
课上习题



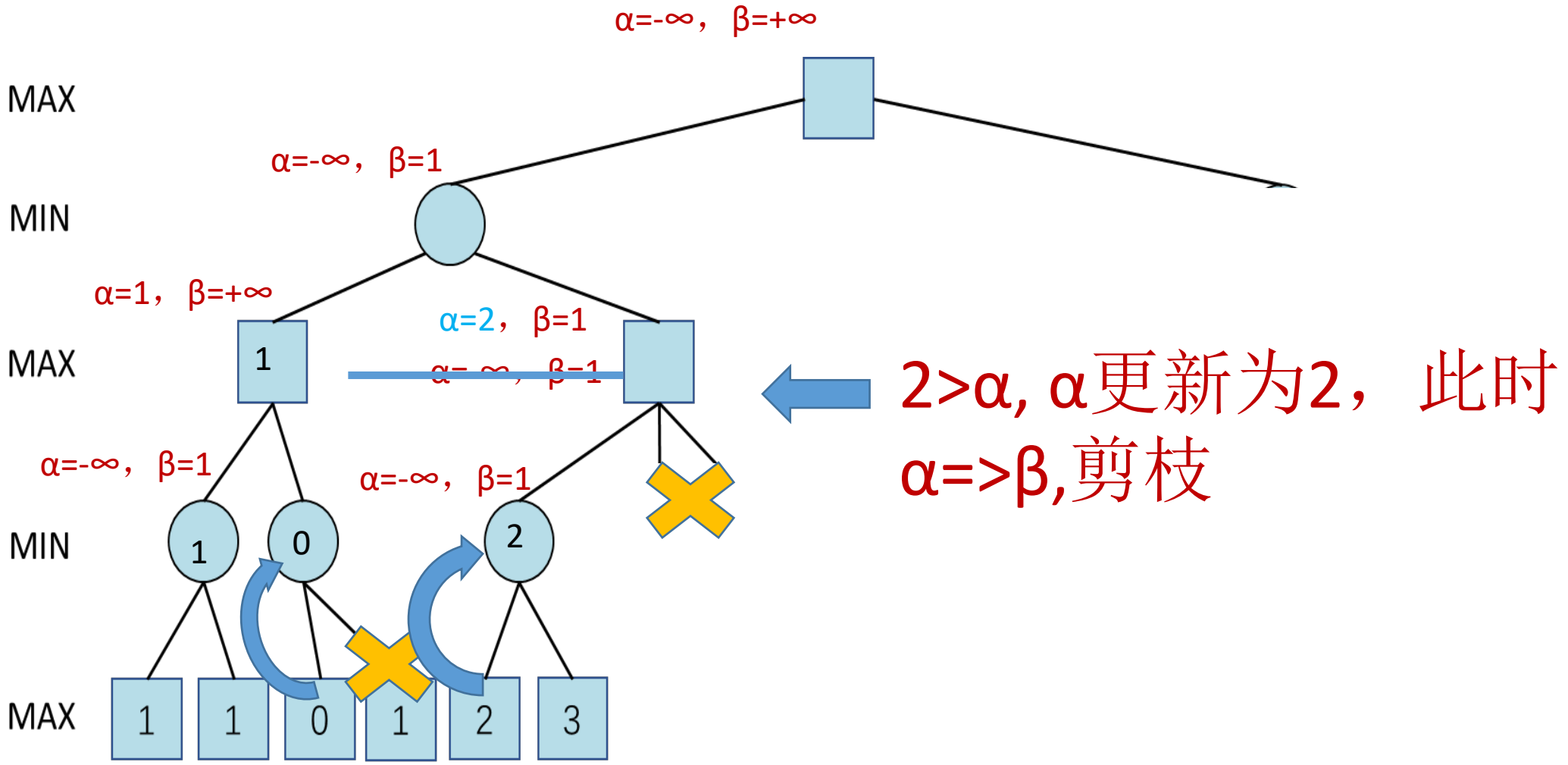
课上习题



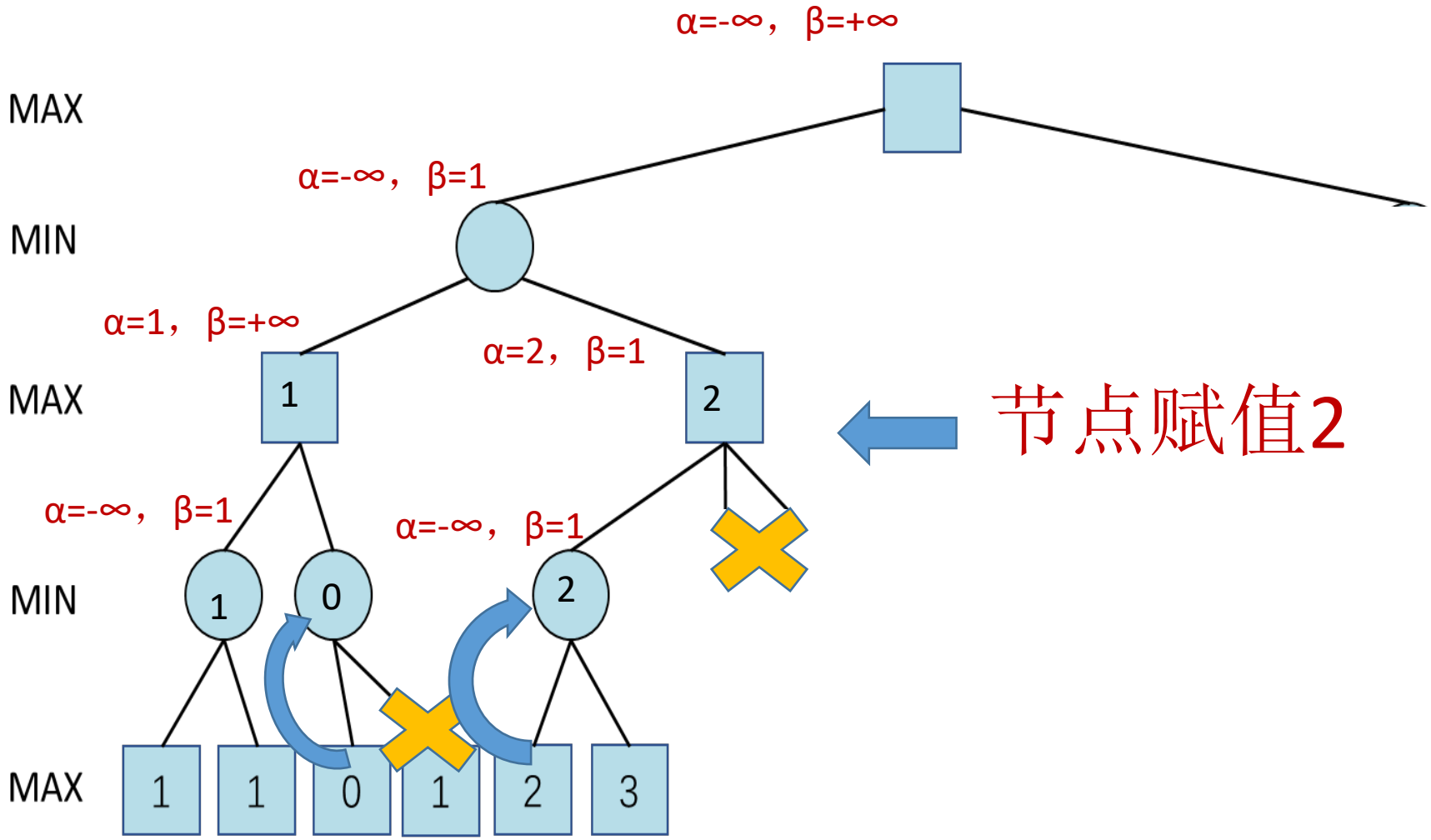
课上习题


α=-∞，β=+∞
MAX
α=-∞，β=1
MIN
α=1，β=+∞
α=-∞，β=1
MAX
1
α=-∞，β=1
α=-∞，β=1
MIN
1
0
2
MAX
1
1
0
1
2
3
全部子节点结束，母节点赋值

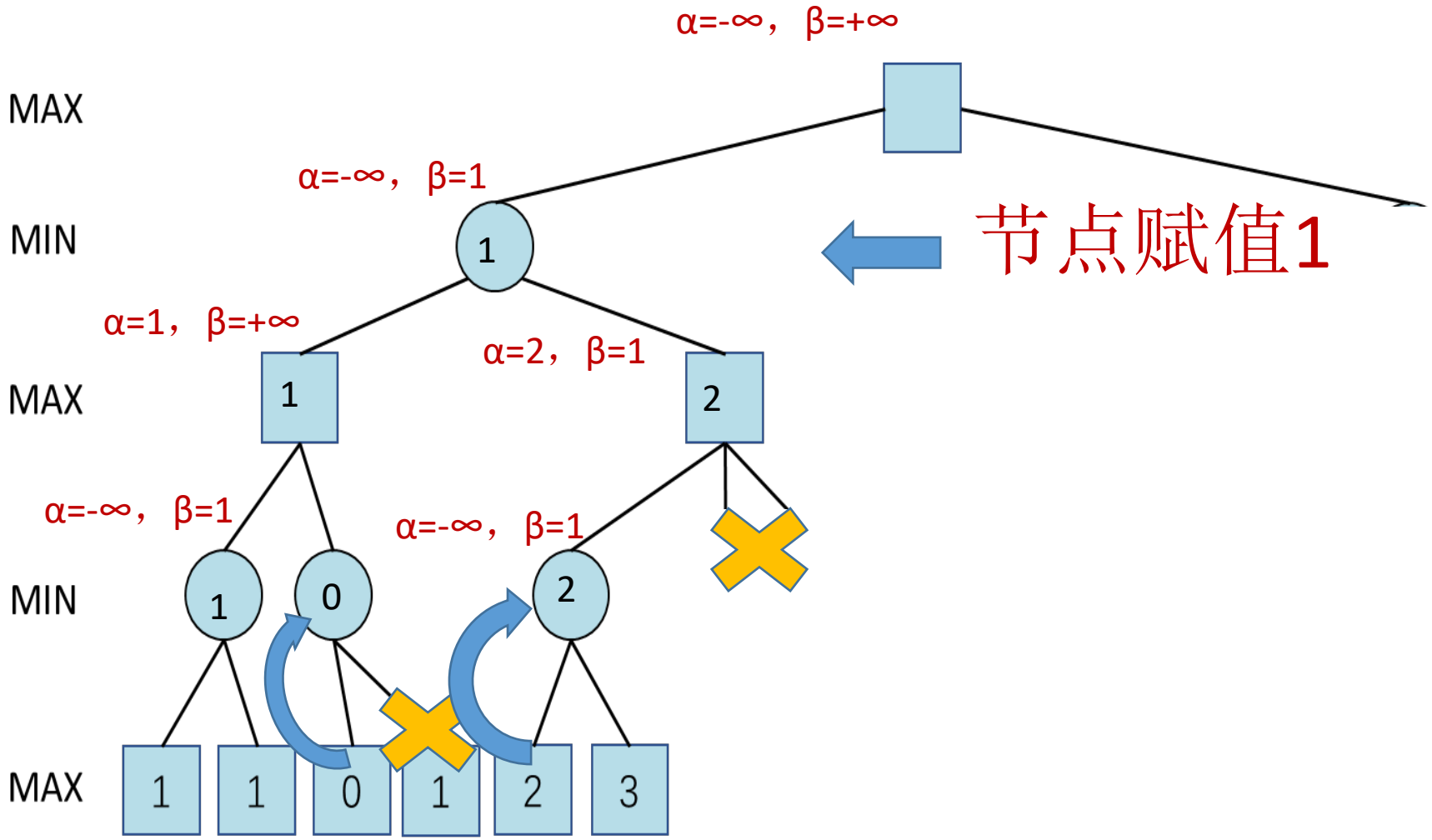
课上习题


α=-∞， β=+∞
MAX
α=-∞， β=1
MIN
α=1， β=+∞
α=2， β=1
MAX
1
α=-∞， β=1
α=-∞， β=1
MIN
1
0
2
MAX
1
1
0
1
2
3
2>α, α更新为2， 此时
α=>β,剪枝

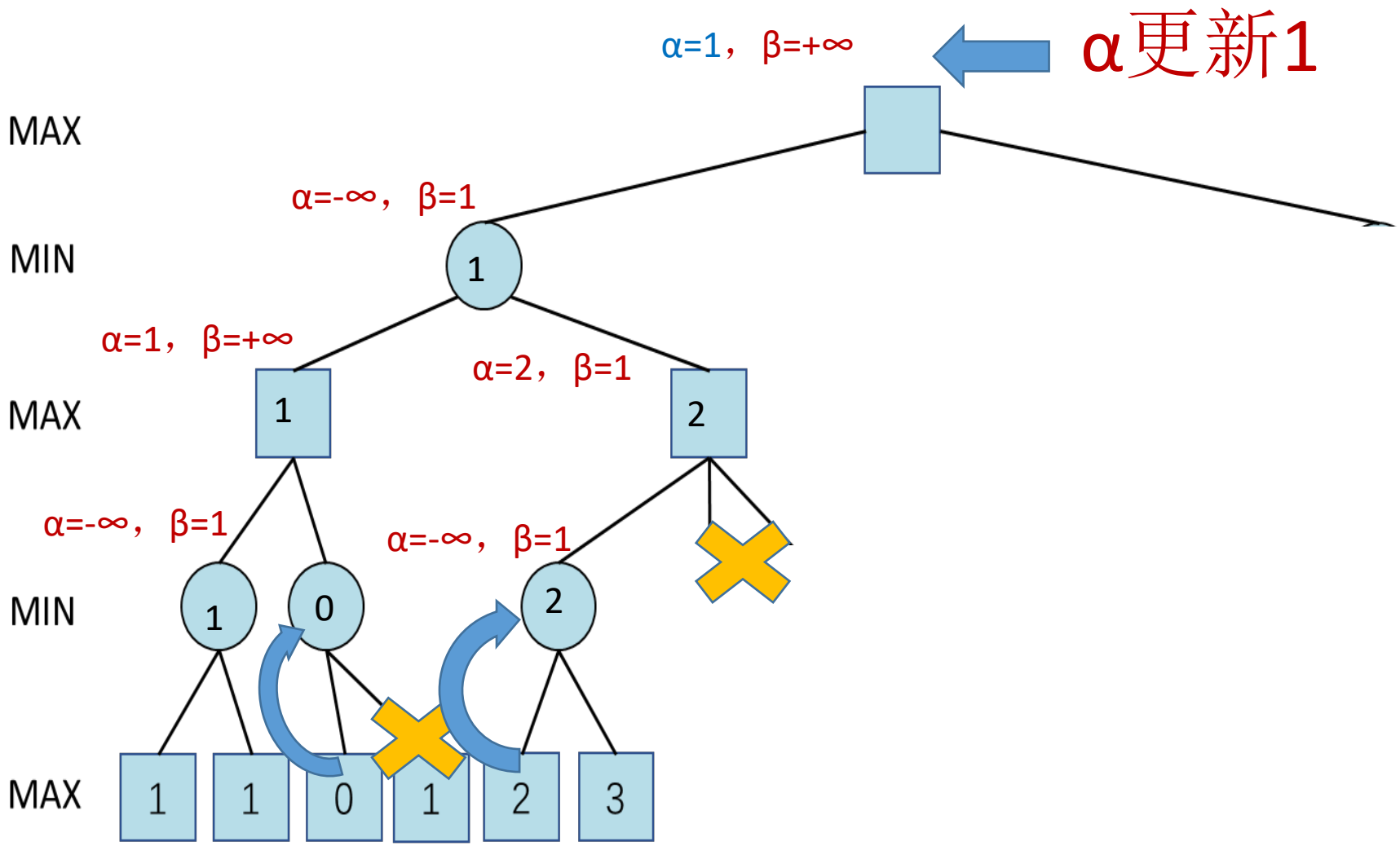
课上习题



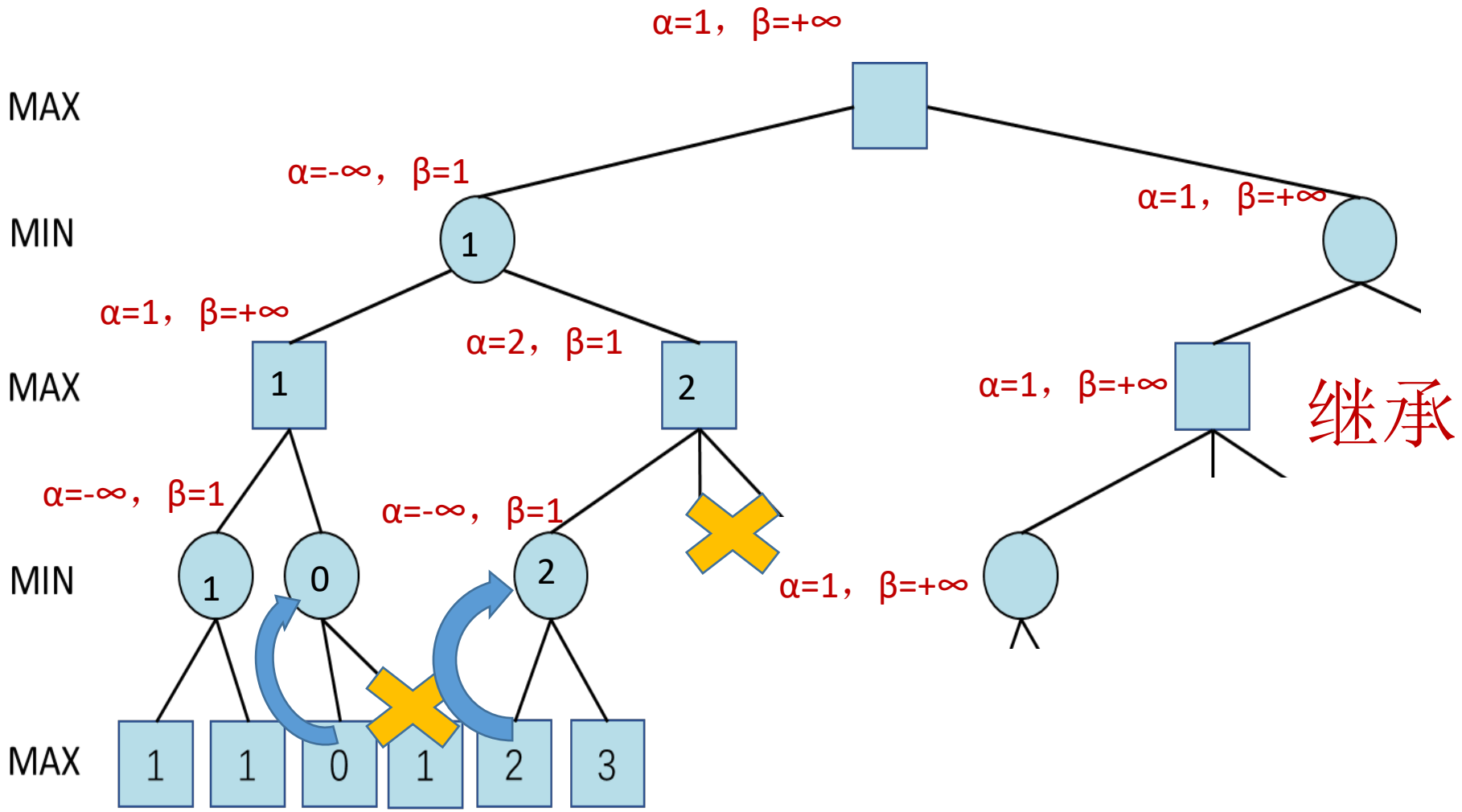
课上习题


α=-∞，β=+∞
MAX
α=-∞，β=1
MIN
1
节点赋值1
α=1，β=+∞
α=2，β=1
MAX
1
2
α=-∞，β=1
α=-∞，β=1
MIN
1
0
2
MAX
1
1
0
1
2
3

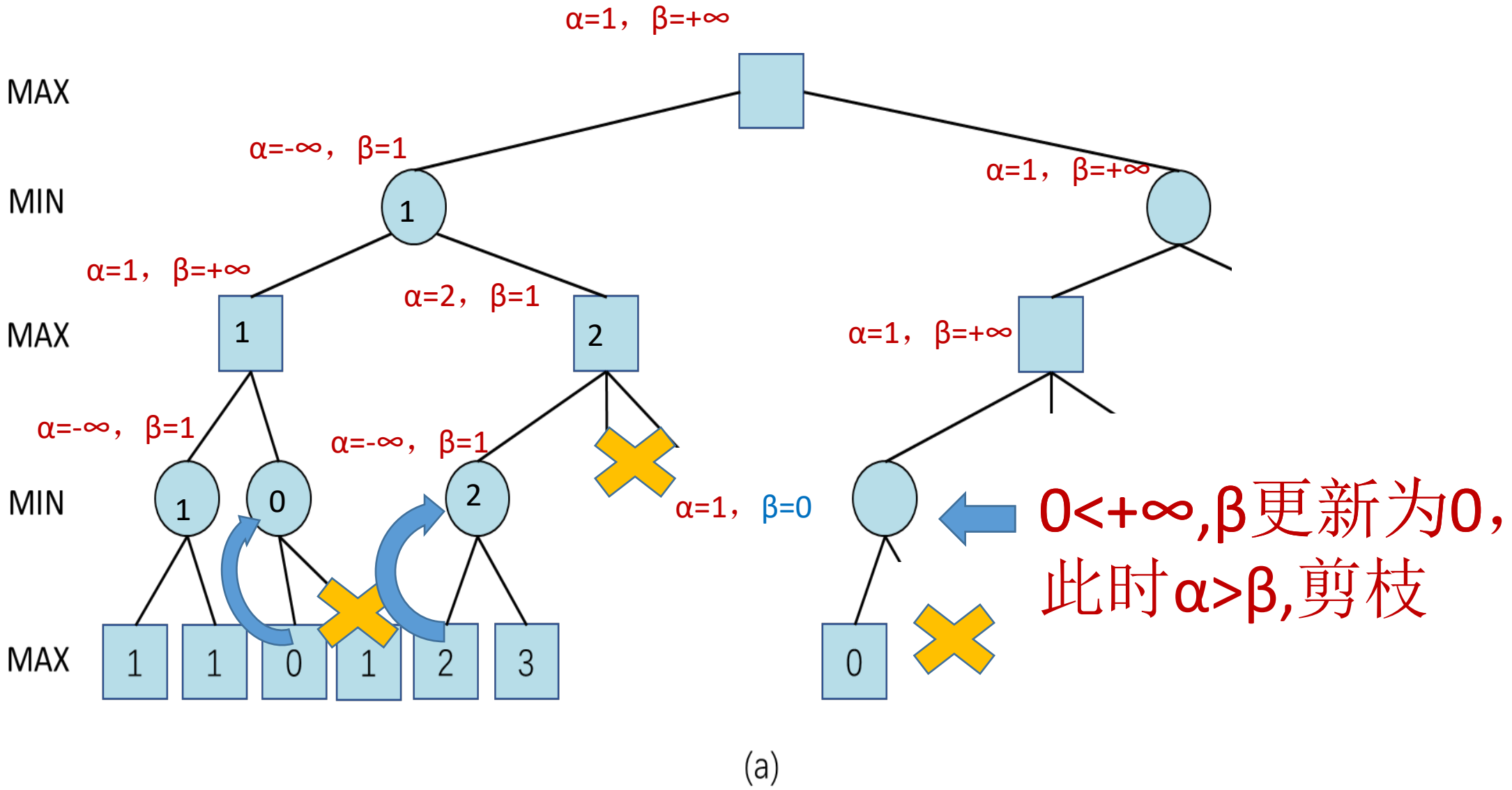
课上习题


α=1，β=+∞
α更新1
MAX
α=-∞，β=1
MIN
1
α=1，β=+∞
α=2，β=1
MAX
1
2
α=-∞，β=1
α=-∞，β=1
MIN
1
0
2
MAX
1
1
0
1
2
3

课上习题

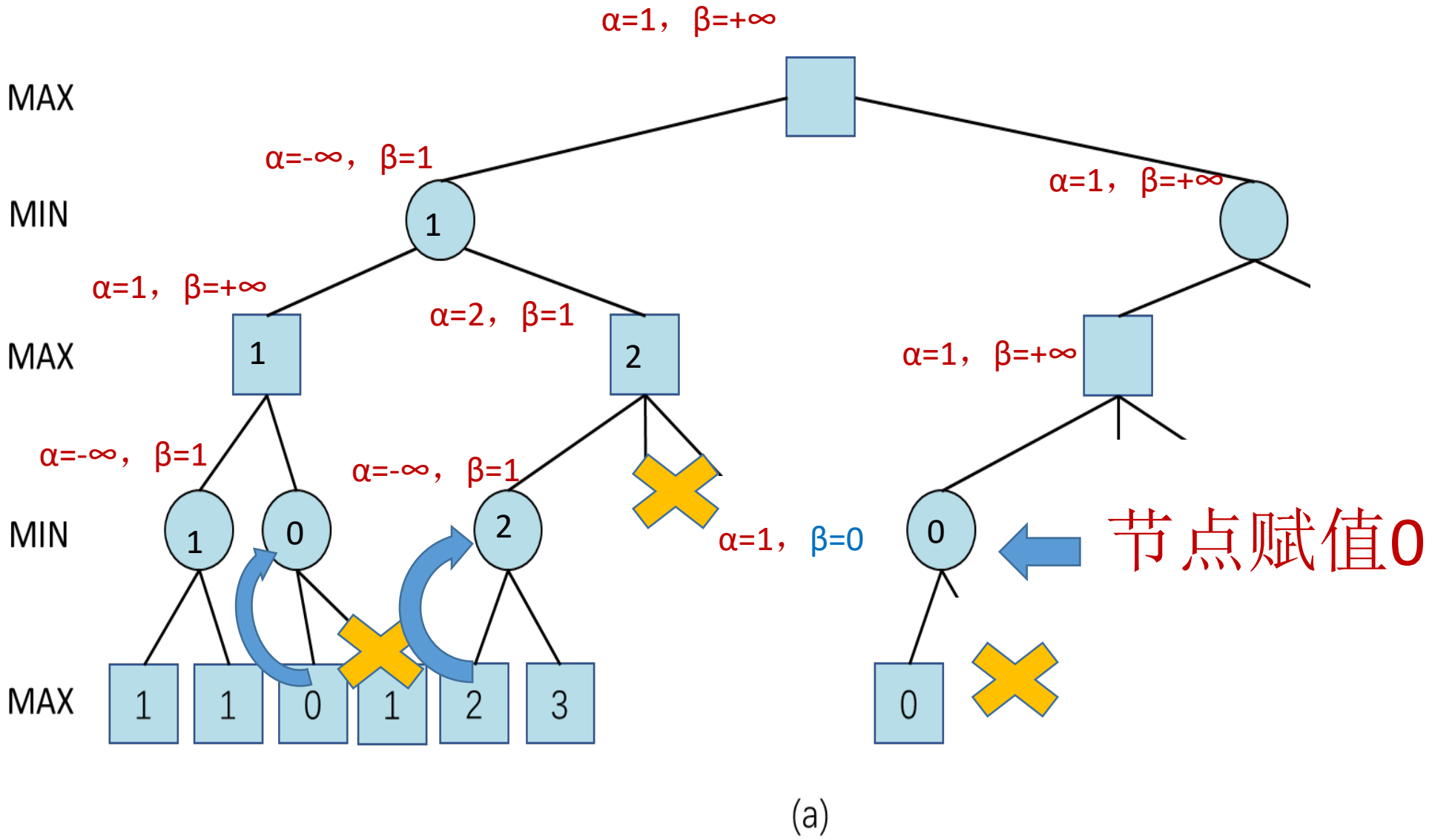

α=1， β=+∞
MAX
α=-∞， β=1
MIN
1
α=1， β=+∞
α=1， β=+∞
α=2， β=1
MAX
1
2
α=1， β=+∞
继承
α=-∞， β=1
α=-∞， β=1
MIN
1
0
2
α=1， β=+∞
MAX
1
1
0
1
2
3

课上习题

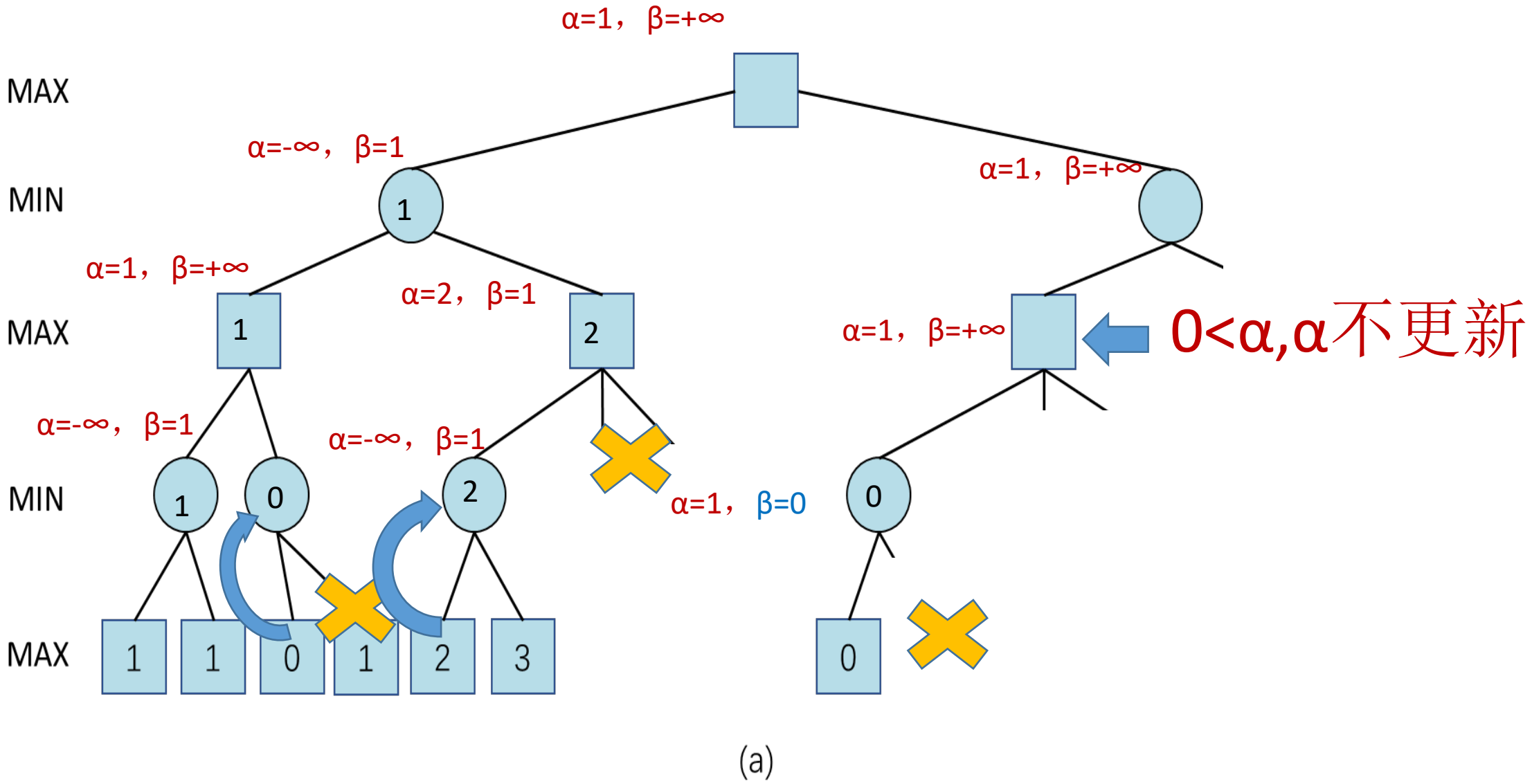

α=1， β=+∞
MAX
α=-∞， β=1
α=1， β=+∞
MIN
1
α=1， β=+∞
α=2， β=1
α=1， β=+∞
MAX
1
2
α=-∞， β=1
α=-∞， β=1
α=1， β=0
MIN
1
0
2
0<+∞,β更新为0，
此时α>β,剪枝
MAX
1
1
0
1
2
3
0

(a)

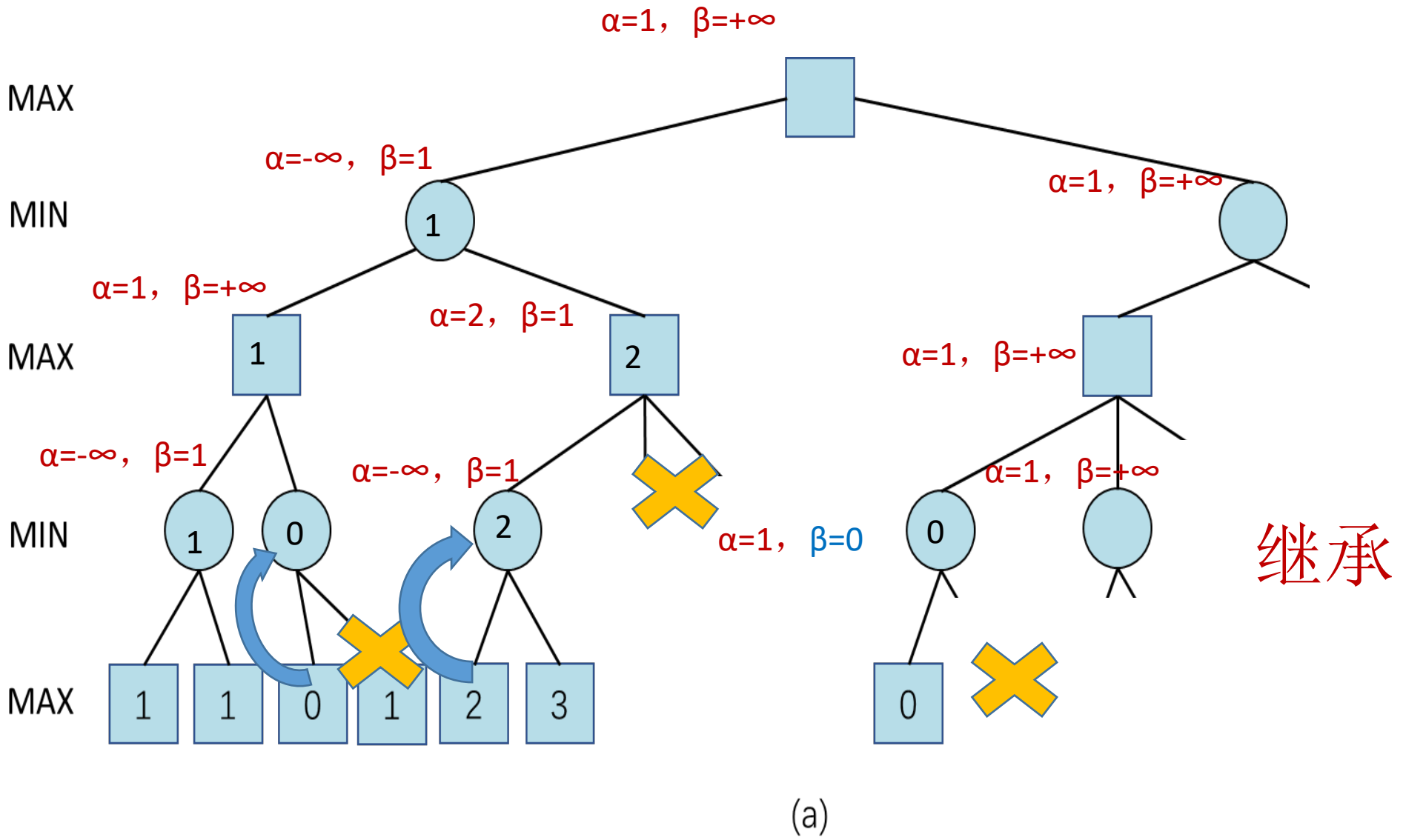
课上习题


α=1， β=+∞
MAX
α=-∞， β=1
MIN
1
α=1， β=+∞
α=1， β=+∞
α=2， β=1
MAX
1
2
α=1， β=+∞
α=-∞， β=1
α=-∞， β=1
MIN
1
0
2
α=1， β=0
0
节点赋值0
MAX
1
1
0
1
2
3
0
(a)

课上习题

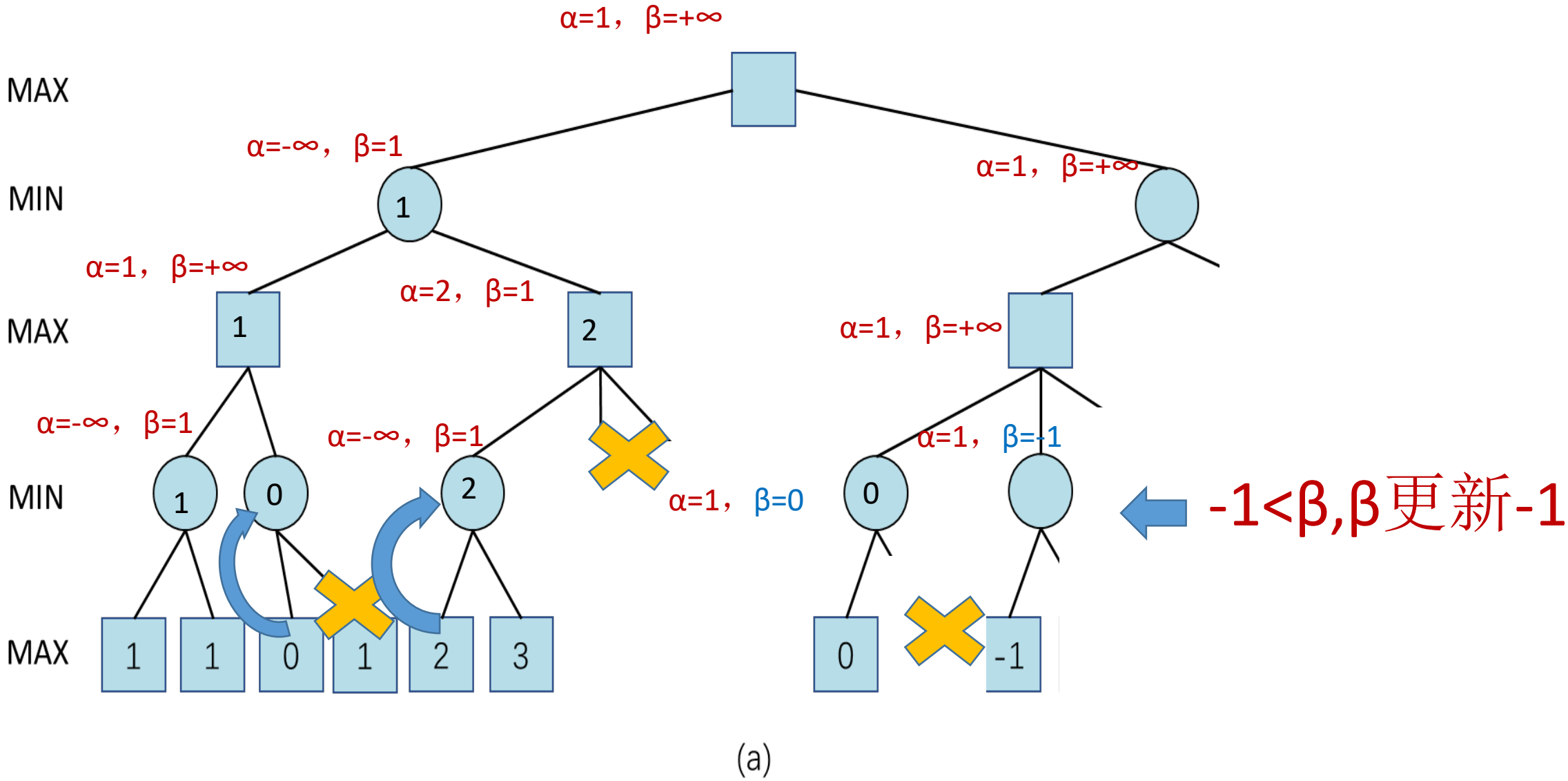

α=1， β=+∞
MAX
α=-∞， β=1
MIN
1
α=1， β=+∞
α=1， β=+∞
α=2， β=1
MAX
1
2
α=1， β=+∞
0<α,α不更新
α=-∞， β=1
α=-∞， β=1
MIN
1
0
2
α=1， β=0
0
MAX
1
1
0
1
2
3
0
(a)

课上习题


α=1， β=+∞
MAX
α=-∞， β=1
1
MIN
α=1， β=+∞
α=1， β=+∞
α=2， β=1
1
2
MAX
α=1， β=+∞
α=-∞， β=1
α=-∞， β=1
α=1， β=+∞
1
0
2
α=1， β=0
0
MIN
继承
1
1
0
1
2
3
0
MAX

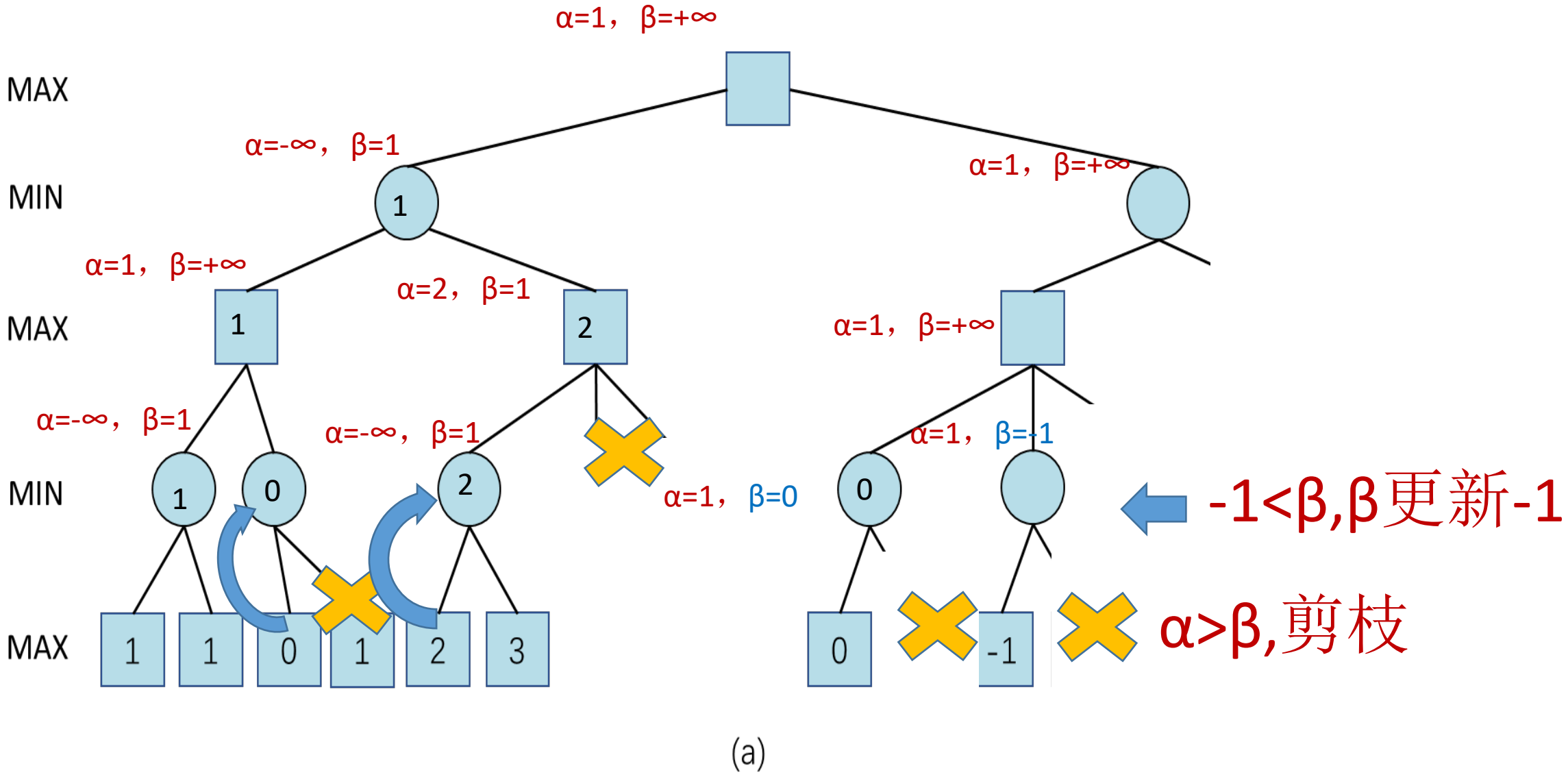
(a)

课上习题


α=1， β=+∞
MAX
α=-∞， β=1
MIN
1
α=1， β=+∞
α=1， β=+∞
α=2， β=1
MAX
1
2
α=1， β=+∞
α=-∞， β=1
α=-∞， β=1
α=1， β=-1
MIN
1
0
2
α=1， β=0
0
-1<β,β更新-1
MAX
1
1
0
1
2
3
0
-1

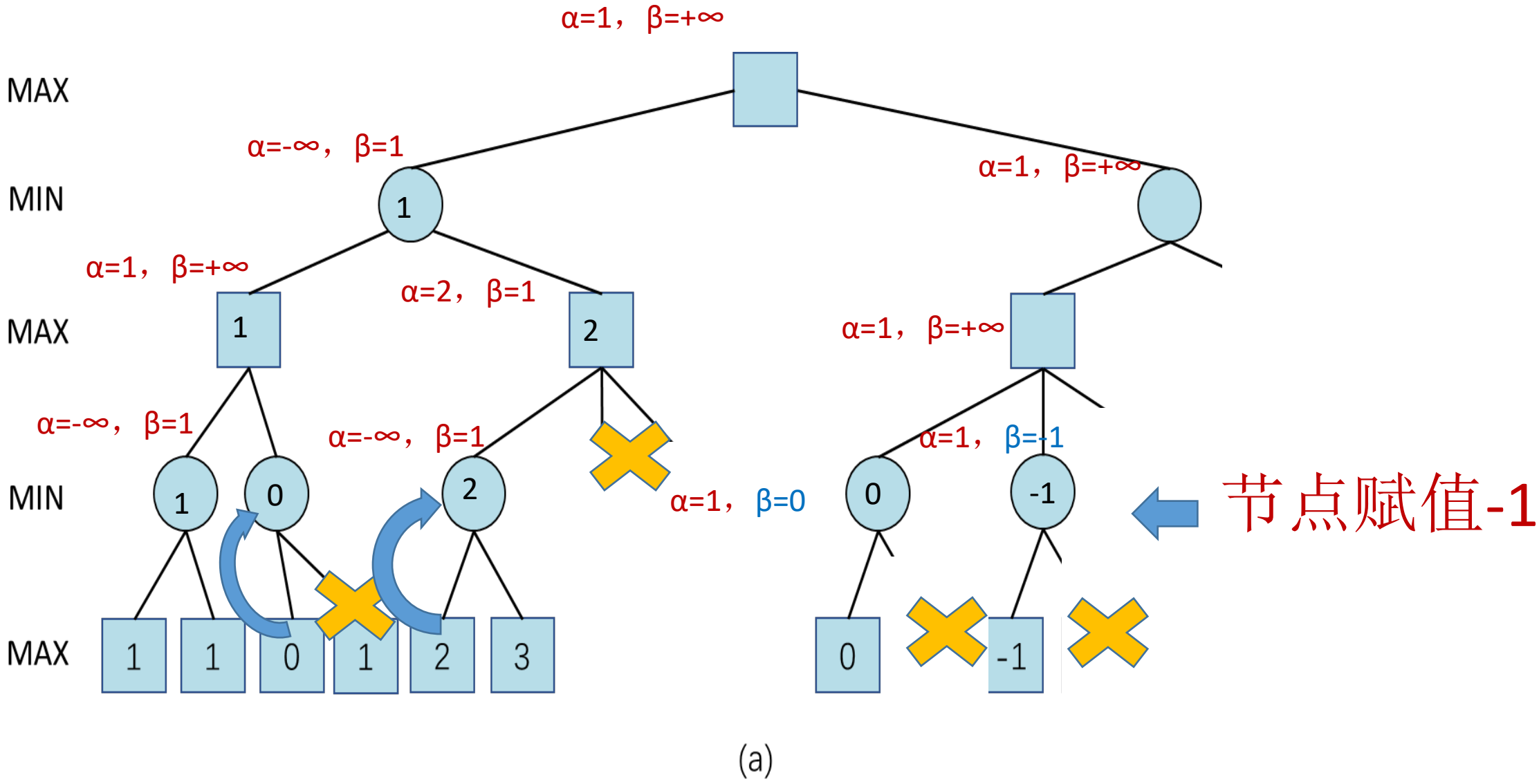
(a)

课上习题

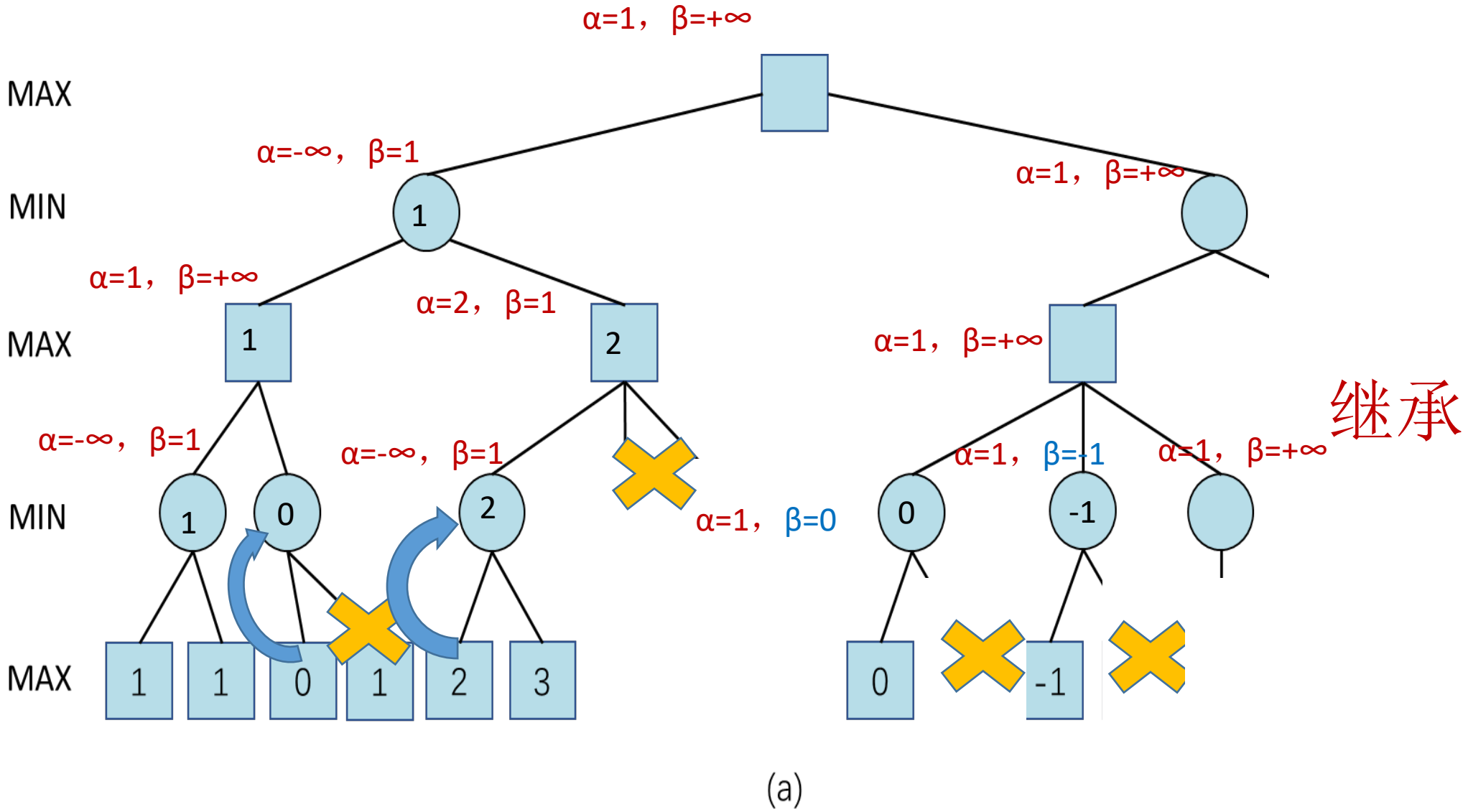

α=1， β=+∞
MAX
α=-∞， β=1
MIN
1
α=1， β=+∞
α=1， β=+∞
α=2， β=1
MAX
1
2
α=1， β=+∞
α=-∞， β=1
α=-∞， β=1
α=1， β=-1
MIN
1
0
2
α=1， β=0
0
-1<β,β更新-1
MAX
1
1
0
1
2
3
0
-1
α>β,剪枝

(a)

课上习题

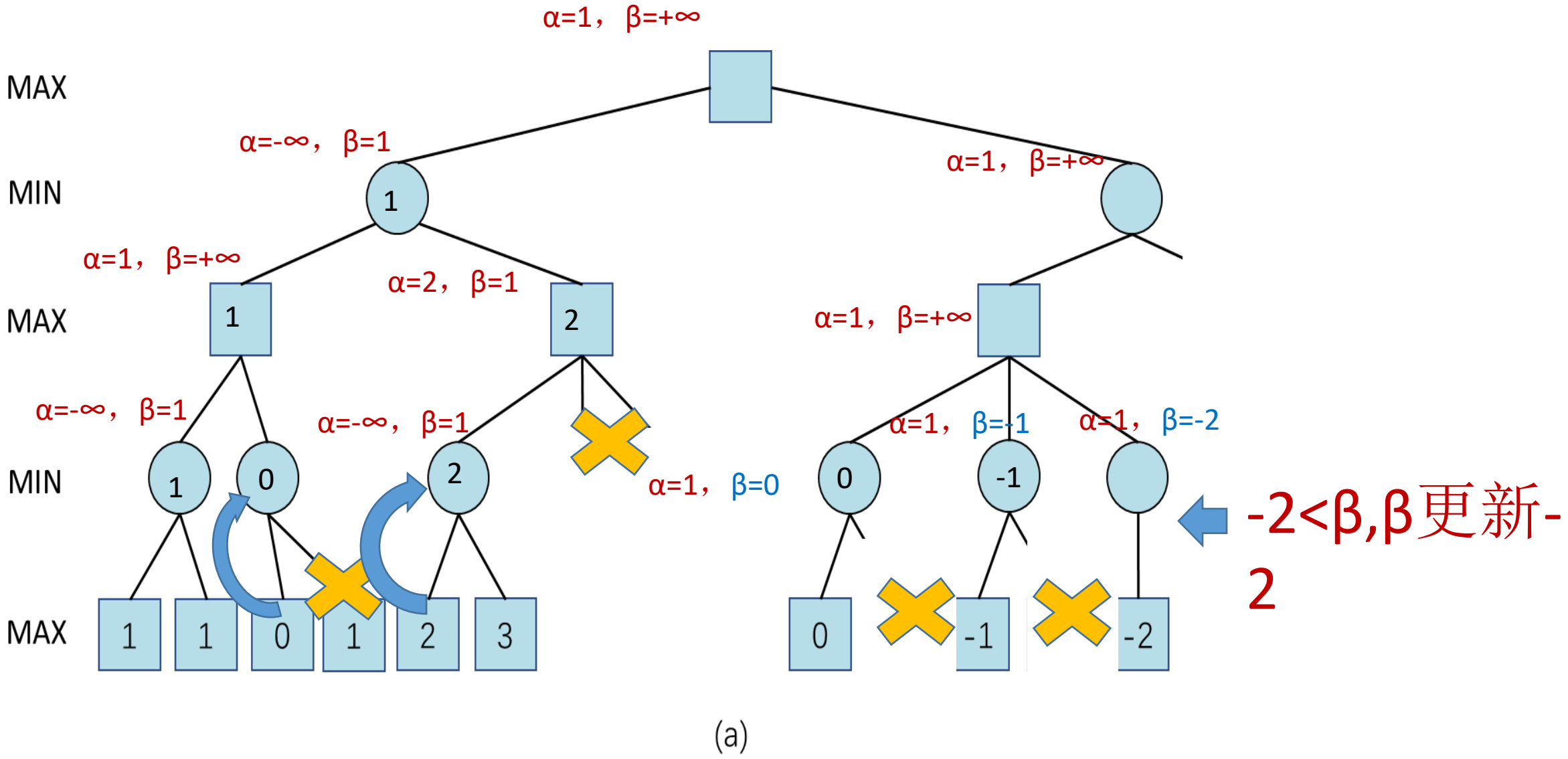

α=1， β=+∞
MAX
α=-∞， β=1
MIN
1
α=1， β=+∞
α=1， β=+∞
α=2， β=1
MAX
1
2
α=1， β=+∞
α=-∞， β=1
α=-∞， β=1
α=1， β=1
MIN
1
0
2
α=1， β=0
0
-1
节点赋值-1
MAX
1
1
0
1
2
3
0
-1
(a)

课上习题


α=1， β=+∞
MAX
α=-∞， β=1
MIN
1
α=1， β=+∞
α=1， β=+∞
α=2， β=1
MAX
1
2
α=1， β=+∞
α=-∞， β=1
α=-∞， β=1
α=1， β=1
α=1， β=+∞
继承
MIN
1
0
2
α=1， β=0
0
-1
MAX
1
1
0
1
2
3
0
-1

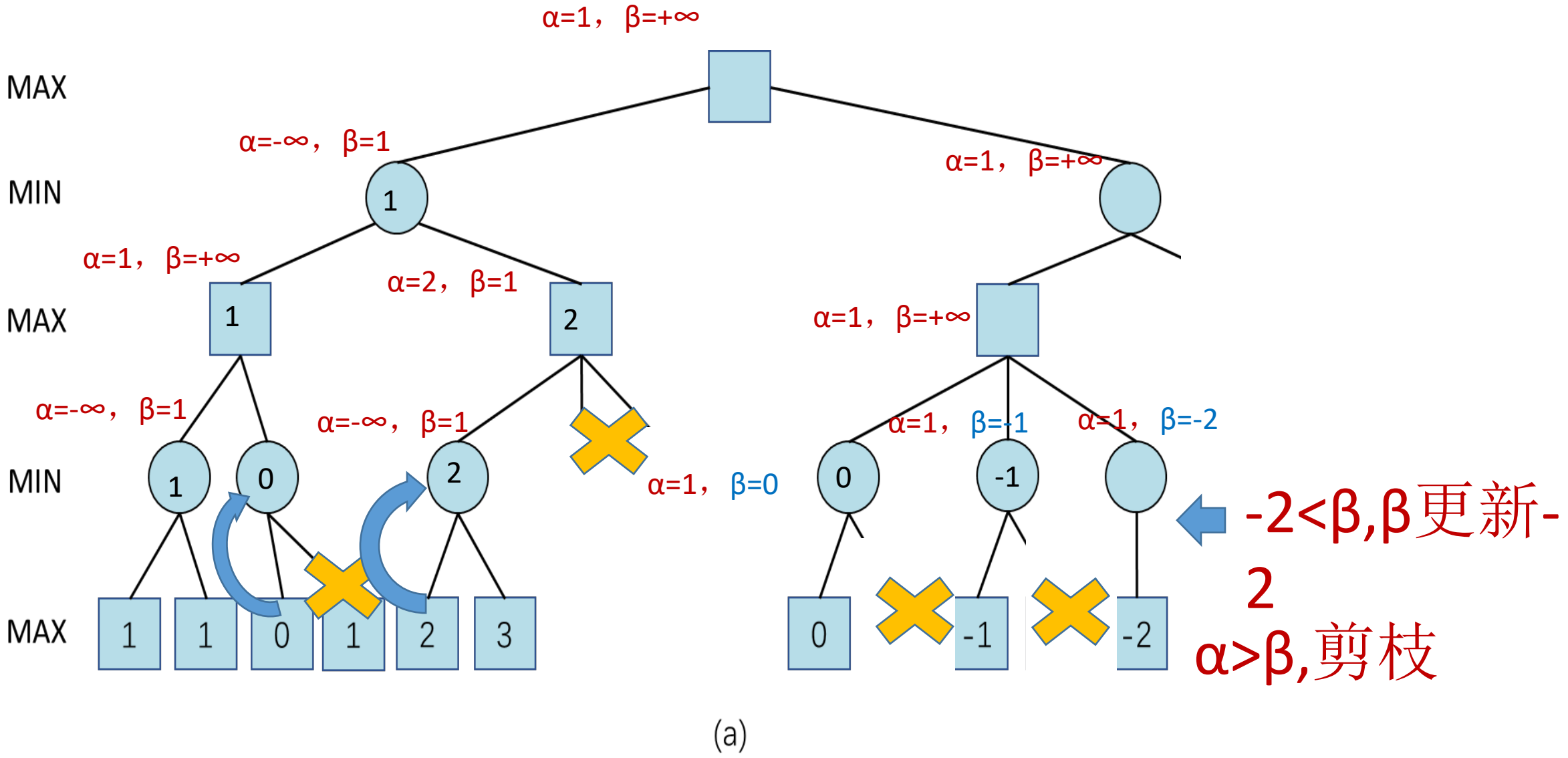
(a)

课上习题


α=1， β=+∞
MAX
α=-∞， β=1
MIN
1
α=1， β=+∞
α=1， β=+∞
α=2， β=1
MAX
1
2
α=1， β=+∞
α=-∞， β=1
α=-∞， β=1
α=1， β=-1
α=1， β=-2
MIN
1
0
2
α=1， β=0
0
-1
-2>β,β更新-2
MAX
1
1
0
1
2
3
0
-1
-2

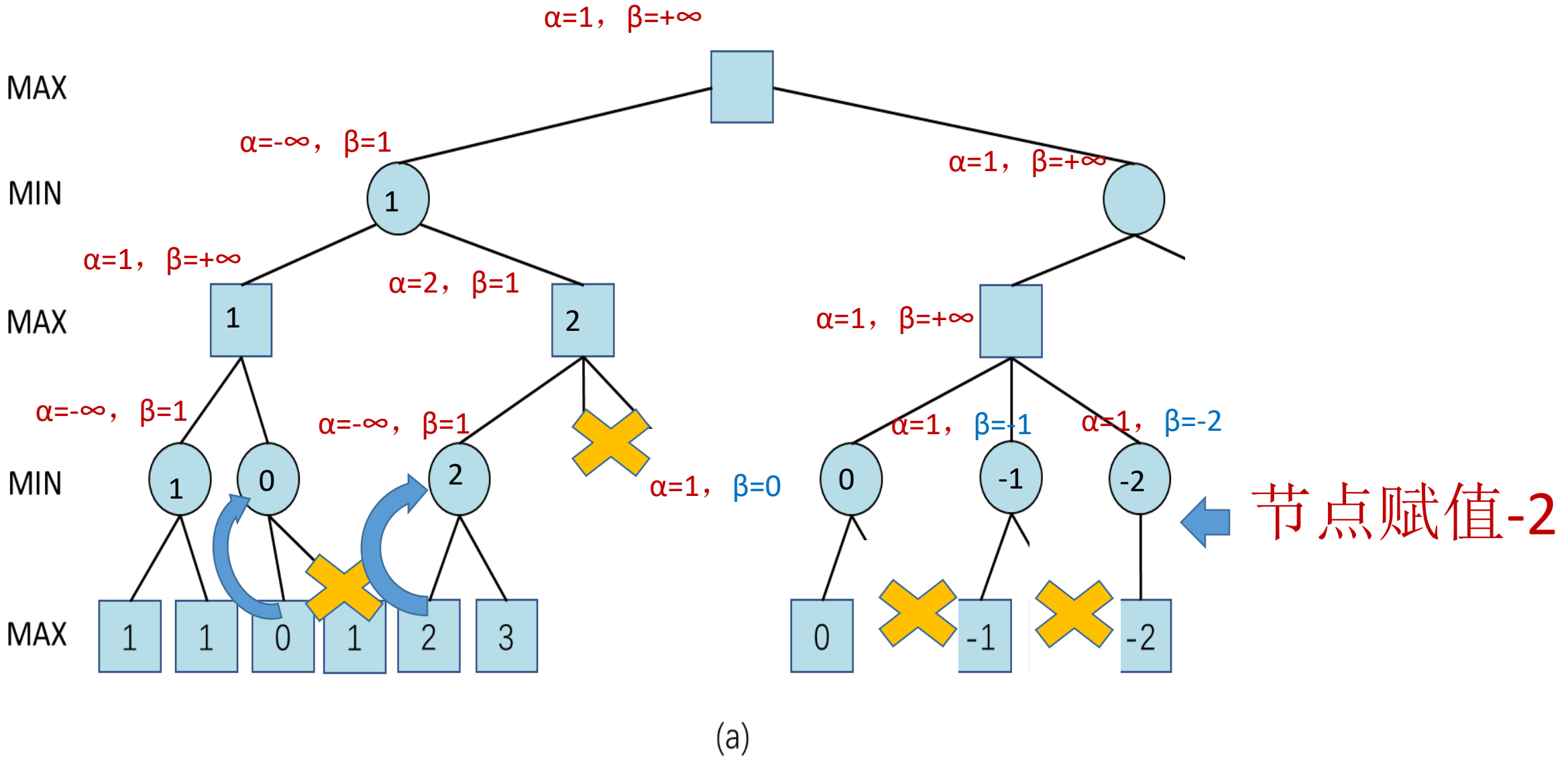
(a)

课上习题


α=1， β=+∞
MAX
α=-∞， β=1
MIN
1
α=1， β=+∞
α=1， β=+∞
α=2， β=1
MAX
1
2
α=1， β=+∞
α=-∞， β=1
α=-∞， β=1
α=1， β=-1
α=1， β=-2
MIN
1
0
2
α=1， β=0
0
-1
1
1
0
1
2
3
0
-1
-2
MAX
-2<β,β更新-2
α>β,剪枝

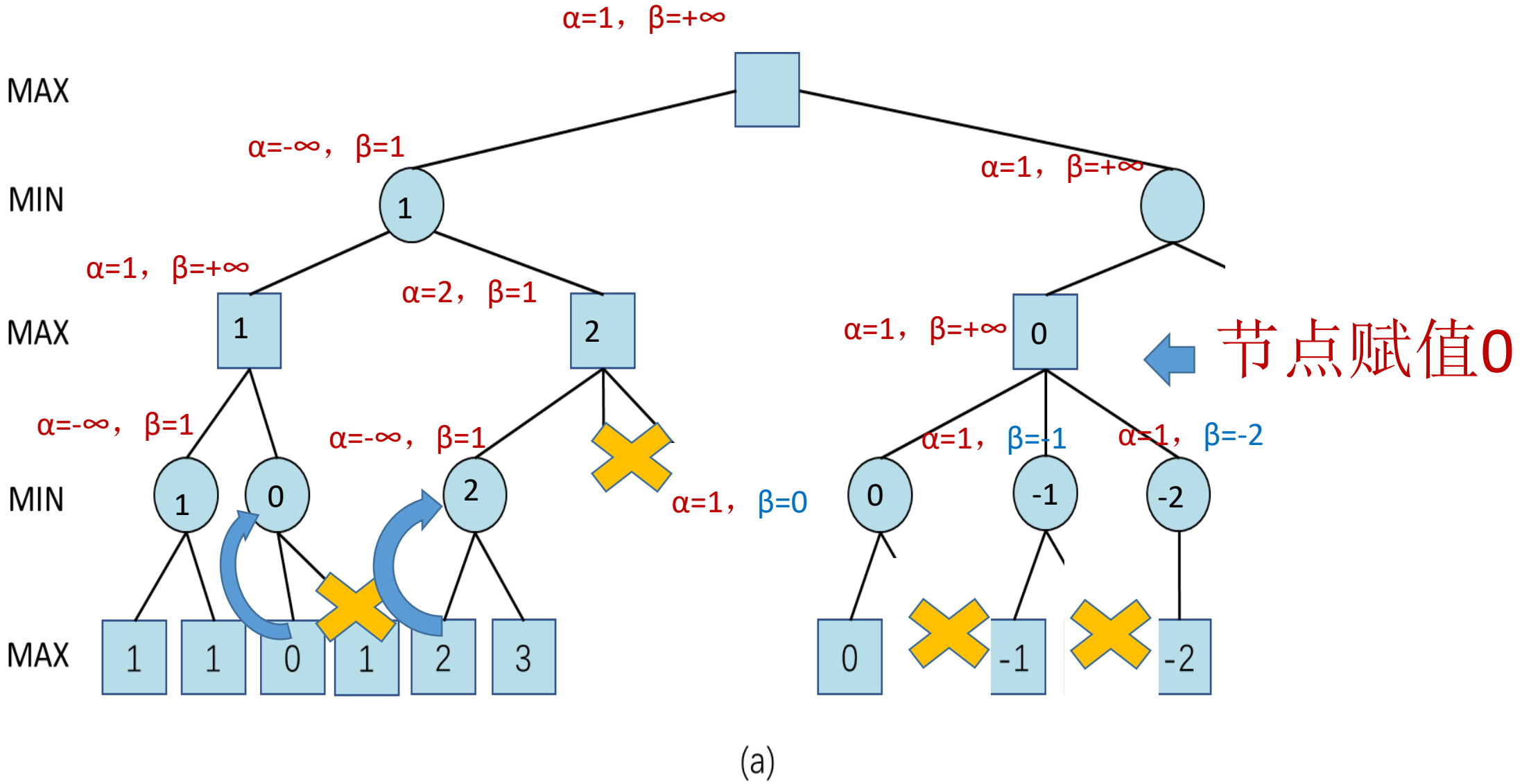
(a)

课上习题


α=1， β=+∞
MAX
α=-∞， β=1
MIN
1
α=1， β=+∞
α=1， β=+∞
α=2， β=1
MAX
1
2
α=1， β=+∞
α=-∞， β=1
α=-∞， β=1
α=1， β=-1
α=1， β=-2
MIN
1
0
2
α=1， β=0
0
-1
-2
节点赋值-2
MAX
1
1
0
1
2
3
0
-1
-2

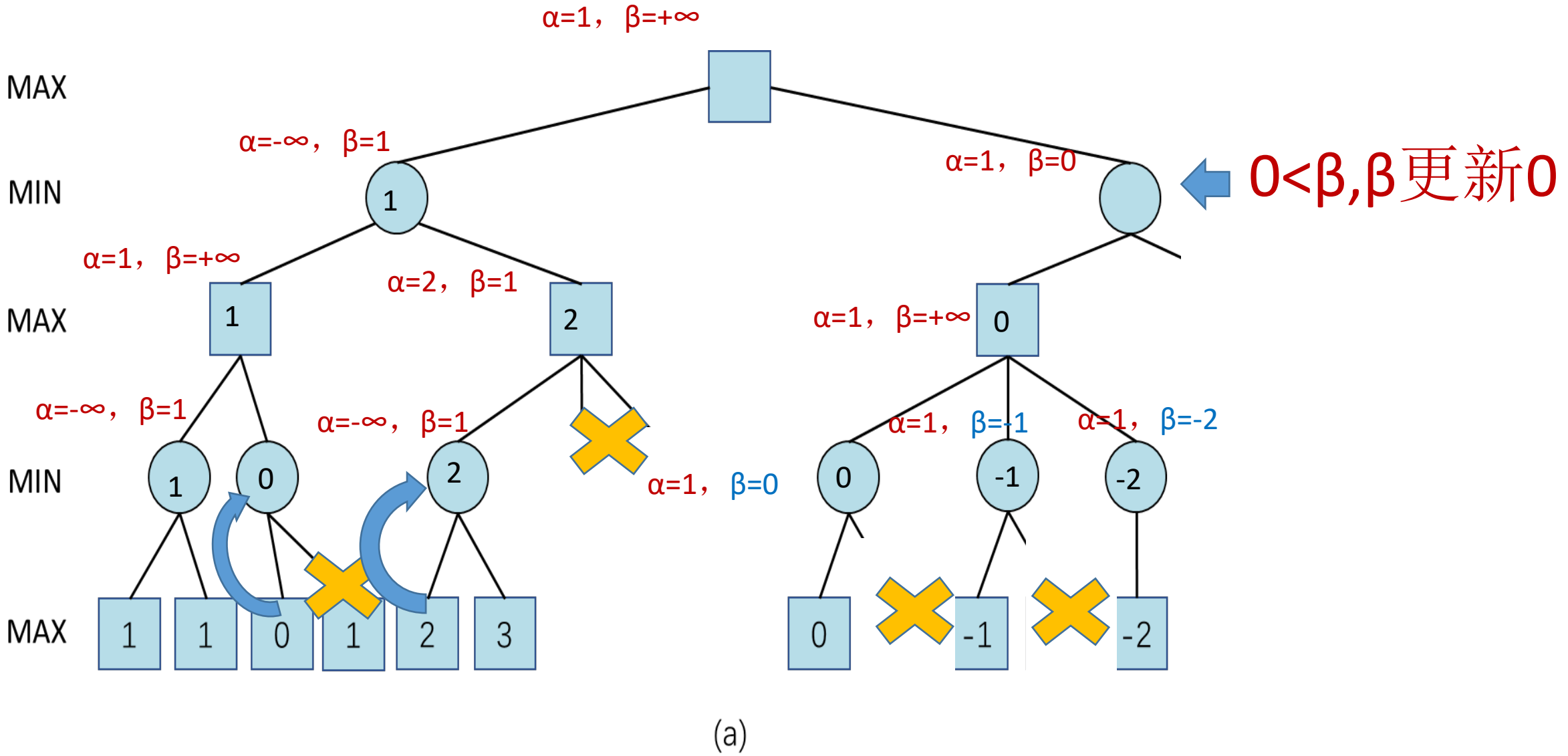
(a)

课上习题

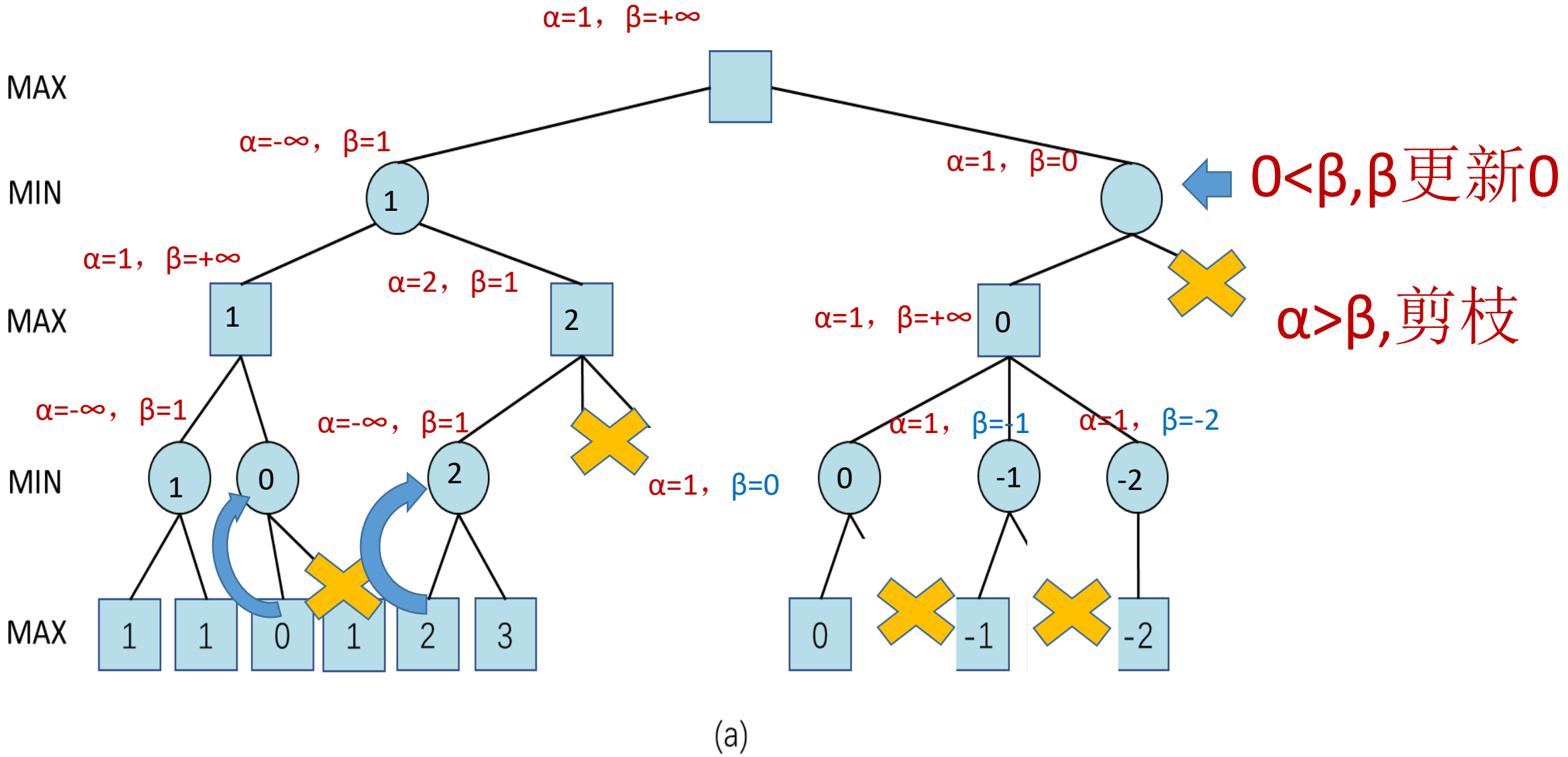

α=1， β=+∞
MAX
α=-∞， β=1
MIN
1
α=1， β=+∞
α=1， β=+∞
α=2， β=1
MAX
1
2
α=1， β=+∞
0
节点赋值0
α=-∞， β=1
α=-∞， β=1
α=1， β=-1
α=1， β=-2
MIN
1
0
2
α=1， β=0
0
-1
-2
MAX
1
1
0
1
2
3
0
-1
-2

(a)

课上习题

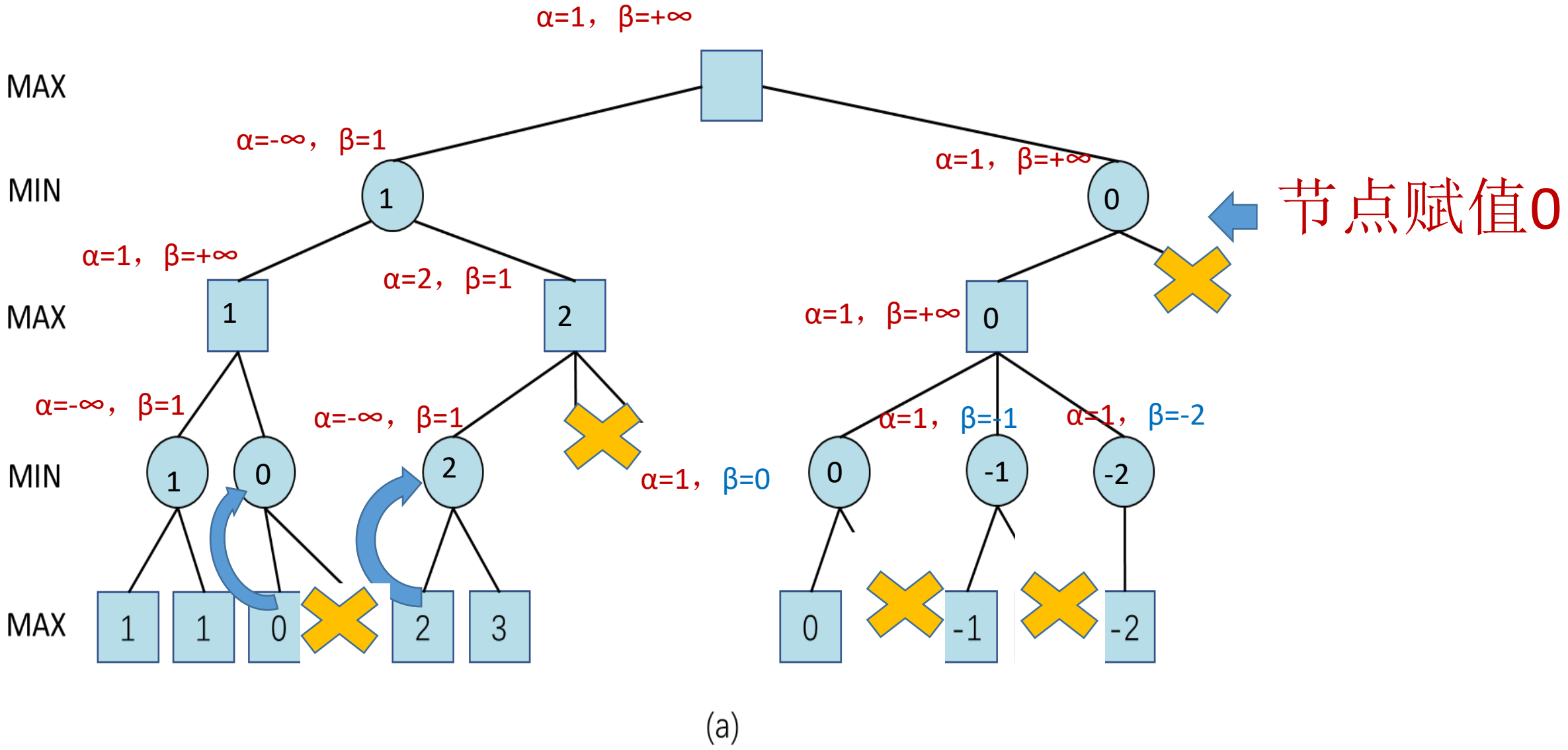

α=1， β=+∞
MAX
α=-∞， β=1
MIN
α=1， β=0
0<β,β更新0
α=1， β=+∞
α=2， β=1
MAX
α=1， β=+∞
α=-∞， β=1
α=-∞， β=1
α=1， β=-1
α=1， β=-2
MIN
α=1， β=0
MAX
(a)

课上习题

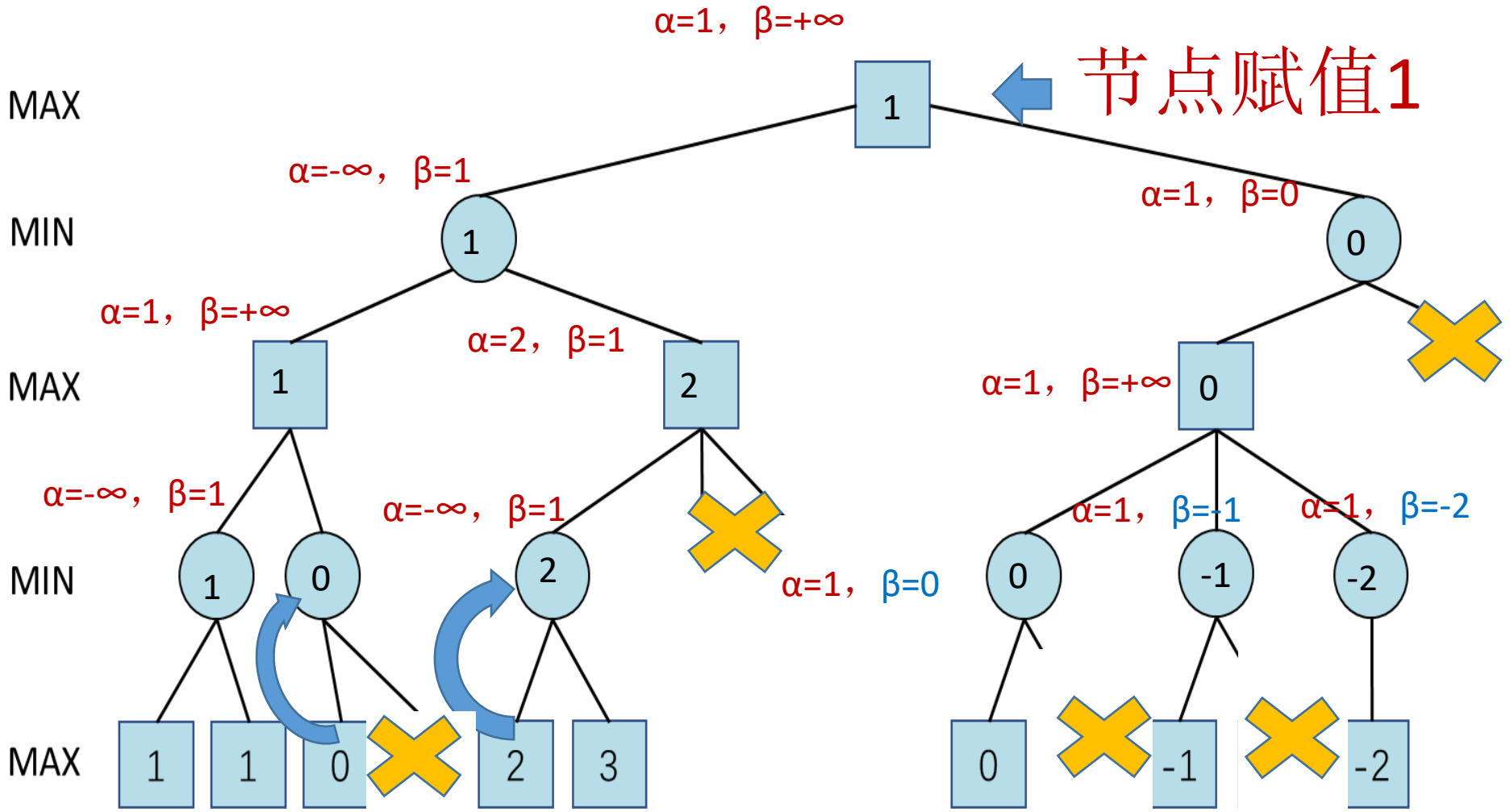

α=1， β=+∞
MAX
α=-∞， β=1
MIN
1
α=1， β=0
0<β,β更新0
α=1， β=+∞
α=2， β=1
MAX
1
2
α=1， β=+∞
0
α>β,剪枝
α=-∞， β=1
α=-∞， β=1
α=1， β=-1
α=1， β=-2
MIN
1
0
2
α=1， β=0
0
-1
-2
MAX
1
1
0
1
2
3
0
-1
-2

(a)

课上习题

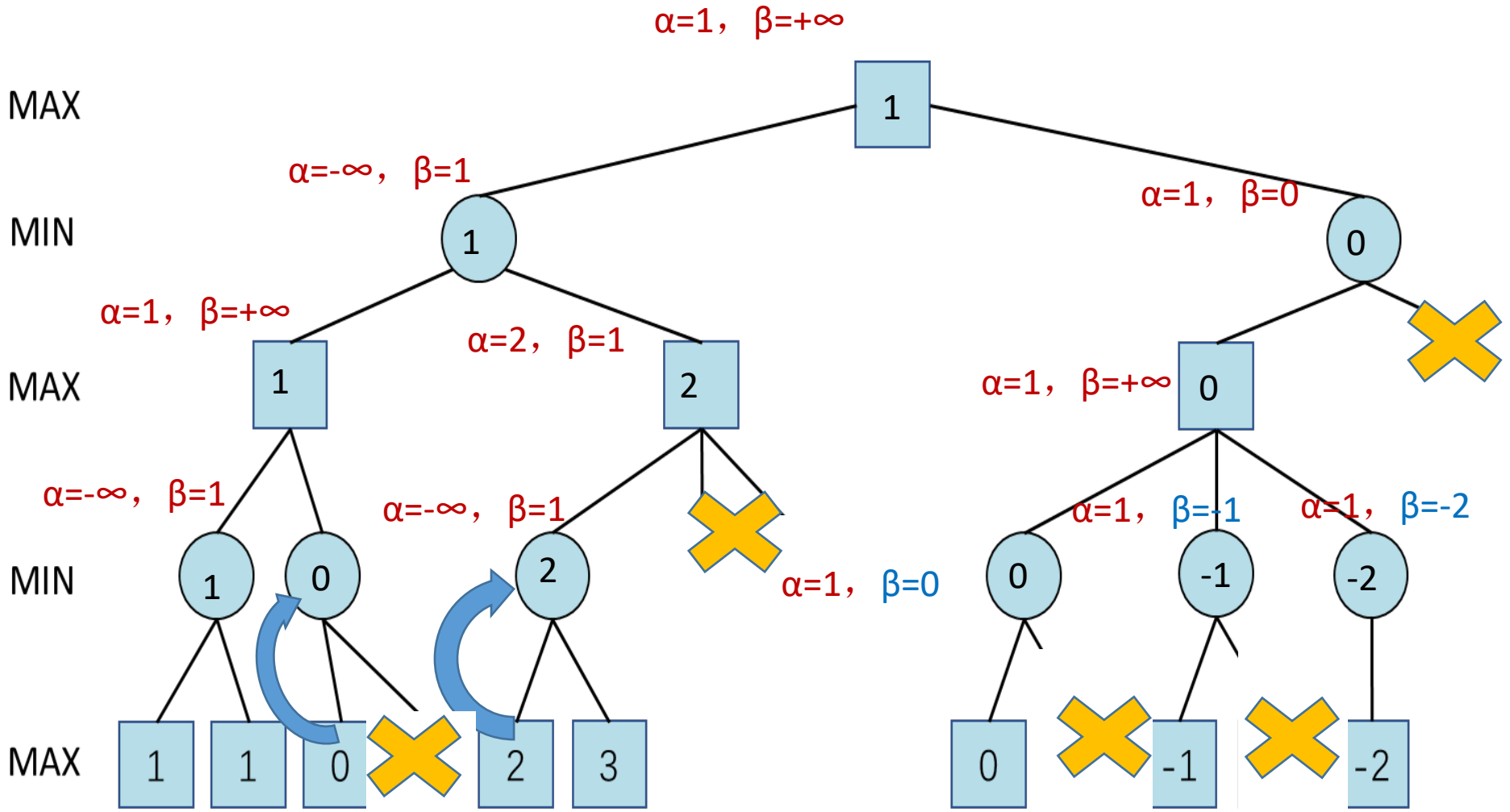

α=1， β=+∞
MAX
α=-∞， β=1
MIN
α=1， β=+∞
α=1， β=+∞
α=2， β=1
MAX
α=1， β=+∞
α=-∞， β=1
α=-∞， β=1
α=1， β=0
α=1， β=-1
α=1， β=-2
MIN
MAX
节点赋值0
(a)

课上习题


α=1， β=+∞
MAX
1
节点赋值1
α=-∞， β=1
α=1， β=0
MIN
1
0
α=1， β=+∞
α=2， β=1
MAX
1
2
α=1， β=+∞
0
α=-∞， β=1
α=-∞， β=1
α=1， β=-1
α=1， β=-2
MIN
1
0
2
α=1， β=0
0
-1
-2
MAX
1
1
0
2
3
0
-1
-2

(a)

课上习题


α=1， β=+∞
MAX
1
α=-∞， β=1
MIN
1
α=1， β=0
0
α=1， β=+∞
MAX
1
α=2， β=1
2
α=1， β=+∞
0
α=-∞， β=1
α=-∞， β=1
α=1， β=0
α=1， β=-1
α=1， β=-2
MIN
1
0
2
0
-1
-2
MAX
1
1
0
2
3
0
-1
-2

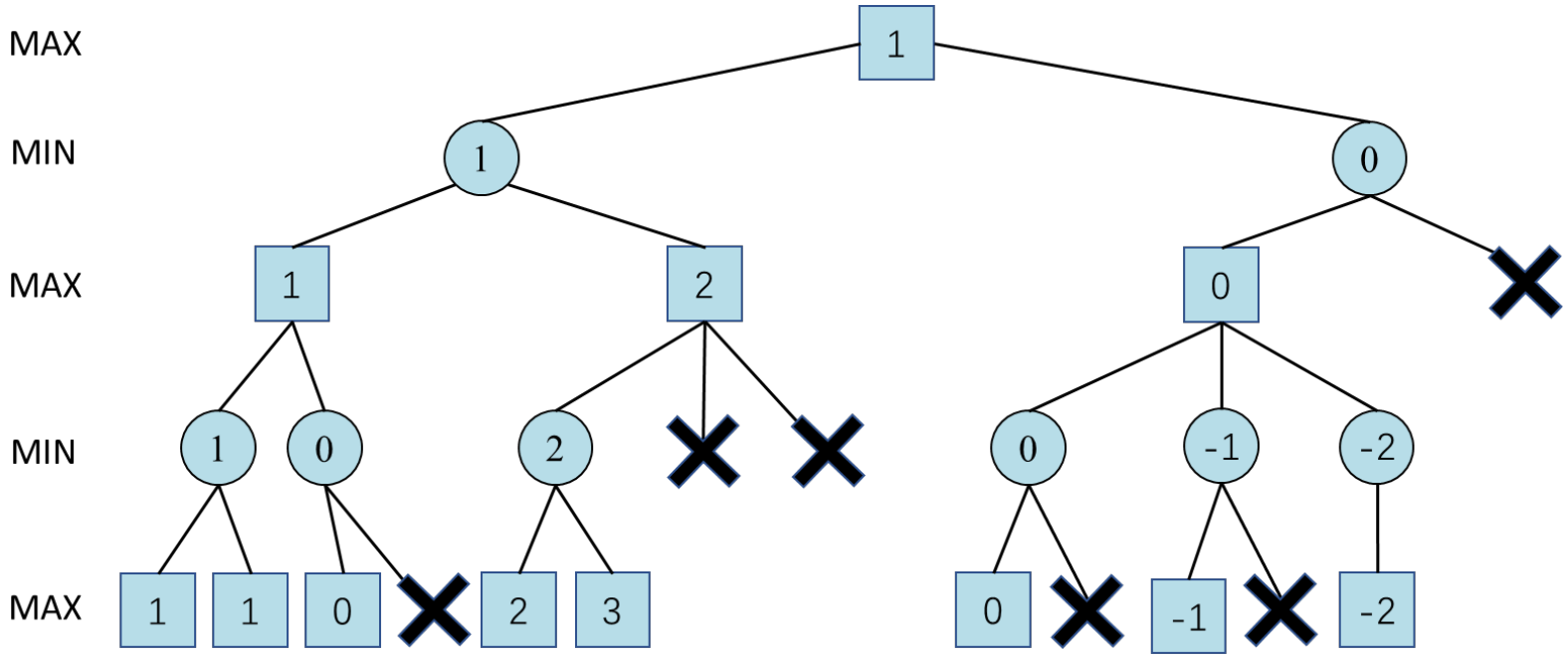
(a)

课上习题

图2展示了一棵Minimax搜索树，可采用alpha-beta剪枝算法进行对抗搜索。假设对于每个节点的后继节点，算法按照从左向右的方向扩展。同时假设当alpha值等于beta值时，算法不进行剪枝。请问：

- (1)对图2(a)中所示搜索树进行搜索，请画出在算法结束时搜索树的状态，用“×”符号标出被剪枝的子树，并计算该算法扩展的节点数量。

  (1) 扩展节点数量为20。

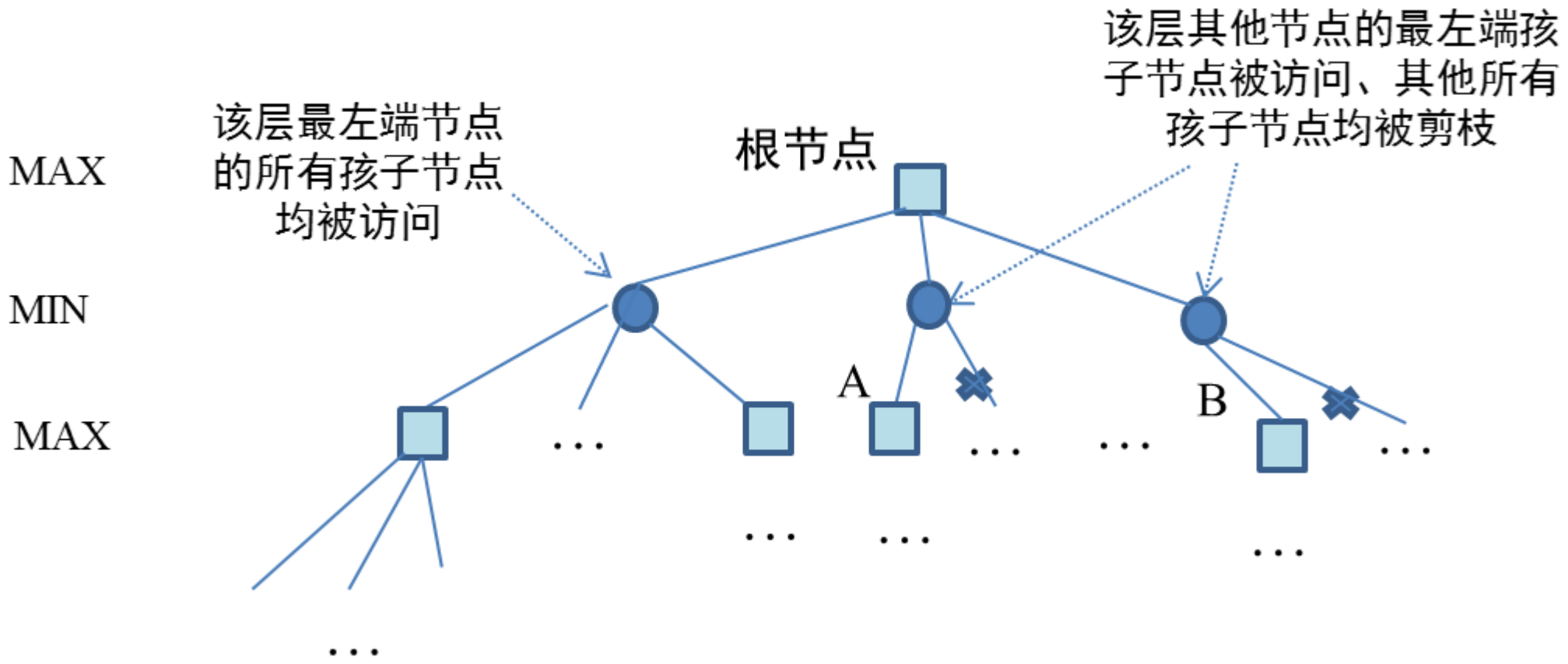


对抗搜索：Alpha-Beta 剪枝搜索的性质

- 剪枝本身不影响算法输出结果
- 节点先后次序会影响剪枝效率
- 如果节点次序“恰到好处”，Alpha-Beta剪枝的时间复杂度为$O(b^{\frac{m}{2}})$，最小最大搜索的时间复杂度为$O(b^m)$

b	分支因子，即搜索树中每个节点最大的分支数目
m	搜索树中路径的最大可能长度

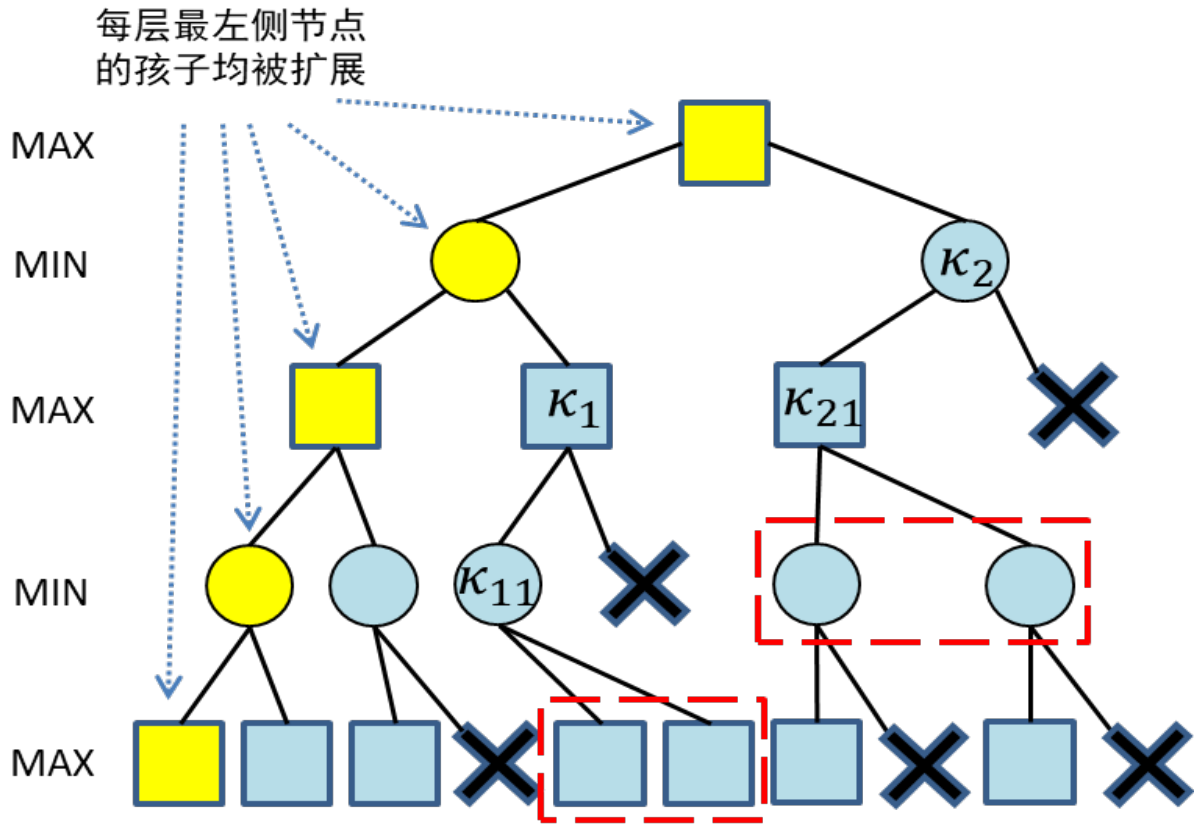
对抗搜索：Alpha-Beta 剪枝算法性能分析



最优状况下的剪枝结果示意图，每一层最左端结点的所有孩子结点均被访问，其他节点仅有最左端孩子结点被访问、其他孩子结点被剪枝。

对抗搜索：Alpha-Beta 剪枝算法性能分析

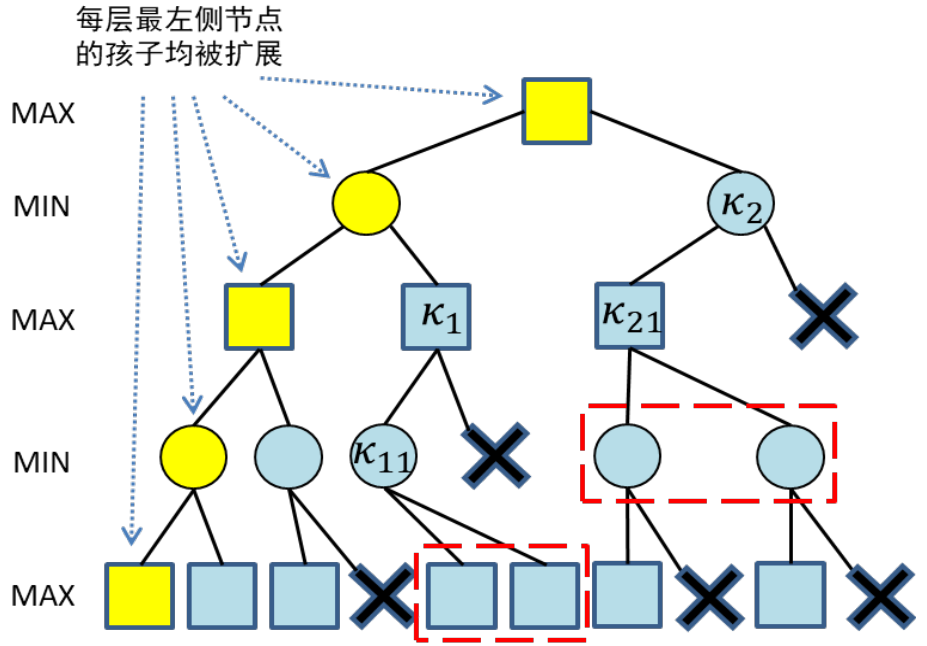
- 观察：如果一个节点导致了其兄弟节点被剪枝，可知其孩子节点必然被扩展



κ_{11}和κ_{21}两个节点分别导致了其右端的兄弟被剪枝，于是可以肯定κ_{11}和κ_{21}全部孩子节点必然被扩展。

对抗搜索：Alpha-Beta 剪枝算法性能分析

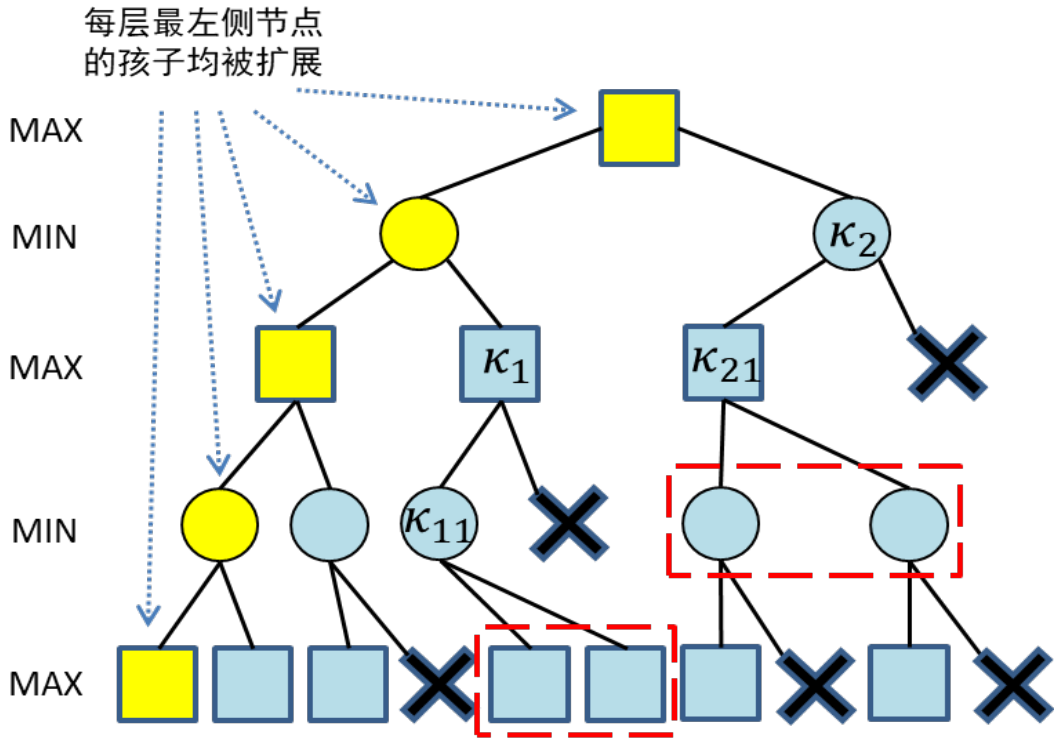
- 下图展示了alpha-beta剪枝算法效率最优的情况
  - 为了方便说明，假设图中每个结点恰好有 b 个子节点
  - 搜索树每层最左端结点的孩子结点必然全部被扩展。其他结点的孩子结点被扩展情况分为如下两类
    - 1)只扩展其最左端孩子结点；
    - 2)它是其父亲结点唯一被扩展的子结点，且它的所有子结点(如果存在)一定被扩展。



对抗搜索：Alpha-Beta 剪枝算法性能分析

- 最左侧子节点外其他子节点
  - 1)只扩展其最左端孩子结点，如κ_1和κ_2；
  - 2)它是其父亲结点唯一被扩展的子结点，且它的所有子结点(如果存在)一定被扩展，如κ_{11}和 κ_{21}

Alpha-Beta剪枝在最优效率下扩展的结点数量为$O(b^{\frac{m}{2}})$，比起最小最大搜索的$O(b^m)$有显著提升。



提纲

- 搜索算法基础
- 启发式搜索
- 对抗搜索
- 蒙特卡洛树搜索

蒙特卡洛树搜索

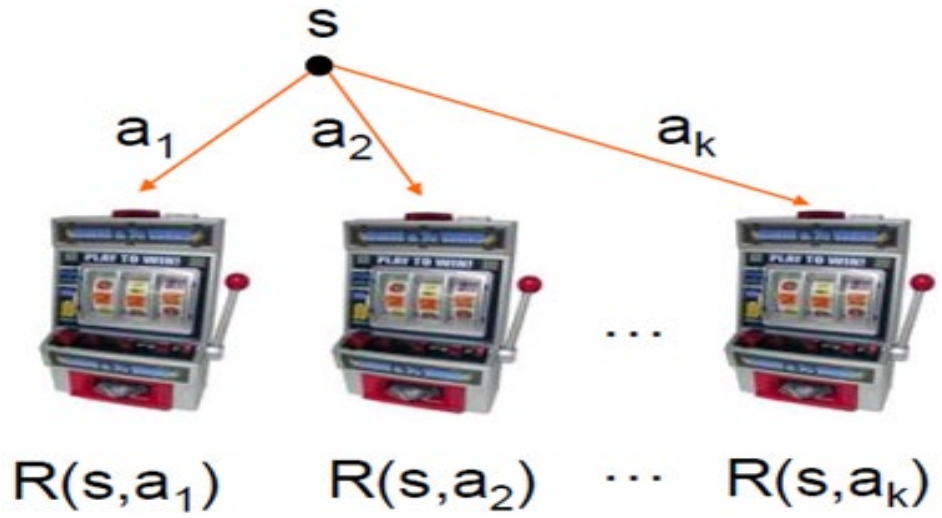
- **推荐阅读材料**
  - D. Silver, et.al., Mastering the game of Go with Deep Neural Networks and Tree Search, Nature, 529:484-490,2016
  - C. Browne, et.al., Survey of Monte Carlo Tree Search Methods, IEEE Transactions on Computational Intelligence and AI in Games, 4(1):1-49,2012
  - S. Gelly, et al., The Grand Challenge of Computer Go: Monte Carlo Tree Search and Extensions, Communications of the ACM, 55(3):106-113,2012
  - L. Kocsis and C. Szepesvari, Bandit Based Monte-Carlo Planning, ECML 2006
  - P. Auer, et. al., Finite-time analysis of the multi-armed bandit problem, Machine learning, 47(2), 235-256, 2002

蒙特卡洛规划 (Monte-Carlo Planning)

- **单一状态蒙特卡洛规划： 多臂赌博机 (multi-armed bandits)**
- **上限置信区间策略 (Upper Confidence Bound Strategies, UCB)**
- **蒙特卡洛树搜索 (Monte-Carlo Tree Search)**
  - UCT (Upper Confidence Bounds on Trees)

单一状态蒙特卡洛规划： 多臂赌博机

- 单一状态， k种行动(即有k 个摇臂)



多臂赌博机

- 多臂赌博机问题是一种序列决策问题，这种问题需要在利用(exploitation)和探索(exploration) 之间保持平衡。

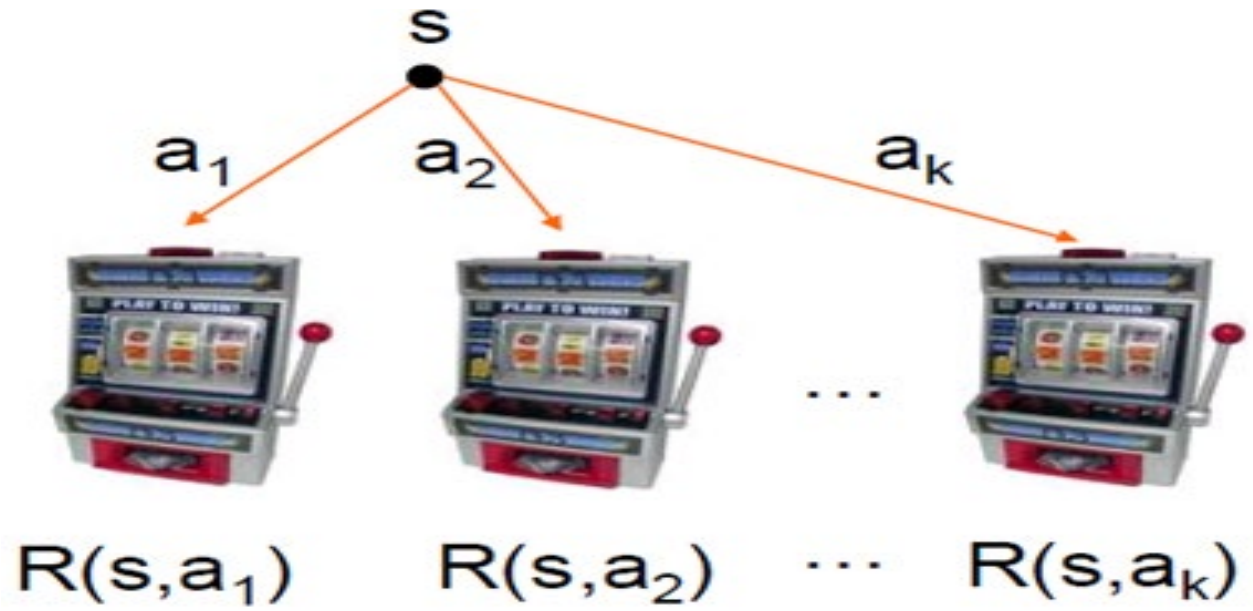
多臂赌博机

$$R_n = \max_{i=1,\ldots,k} \sum_{t=1}^{n} X_{i,t} - \sum_{t=1}^{n} X_{I_t,t}$$

- 悔值函数表示了在第t次对赌博机操作时，假设知道哪个赌博机能够给出最大奖赏(虽然现实中不存在)，则将得到的最大奖赏减去实际操作第I_t个赌博机所得到的奖赏。将n次操作的差值累加起来，就是悔值函数的结果。
- 很显然，一个良好的多臂赌博机操作的策略是在不同人进行了多次玩法后，能够让悔值函数的期望最小。

ϵ −贪心算法

如何奖励最大



上限置信区间 (Upper Confidence Bound, UCB)

- 在多臂赌博机的研究过程中，上限置信区间成为一种较为成功的策略学习方法，因为其在探索-利用之间取得平衡。

上限置信区间 (Upper Confidence Bound, UCB)

- **也就是说，在第t时刻，UCB算法一般会选择具有如下最大值的第j个赌博机：**

$$UCB = \bar{X}_j + \sqrt{\frac{2\ln n}{n_j}} \text{ 或者 } UCB = \bar{X}_j + C \times \sqrt{\frac{2\ln n}{n_j}}$$

$\bar{X}_j$是第j个赌博机在过去时间内所获得的平均奖赏值，n_j是在过去时间内拉动第j个赌博机臂膀的总次数，n是过去时间内拉动所有赌博机臂膀的总次数。C是一个平衡因子，其决定着在选择时偏重探索还是利用。

上限置信区间 (Upper Confidence Bound, UCB)

- **也就是说，在第t时刻，UCB算法一般会选择具有如下最大值的第j个赌博机：**

$$UCB = \bar{X}_j + \sqrt{\frac{2\ln n}{n_j}} \quad \textbf{或者} \quad UCB = \bar{X}_j + C \times \sqrt{\frac{2\ln n}{n_j}}$$

从这里可看出UCB算法如何在探索-利用之间寻找平衡：既需要拉动在过去时间内获得最大平均奖赏的赌博机，又希望去选择那些拉动臂膀次数最少的赌博机。

上限置信区间

- **UCB算法描述**

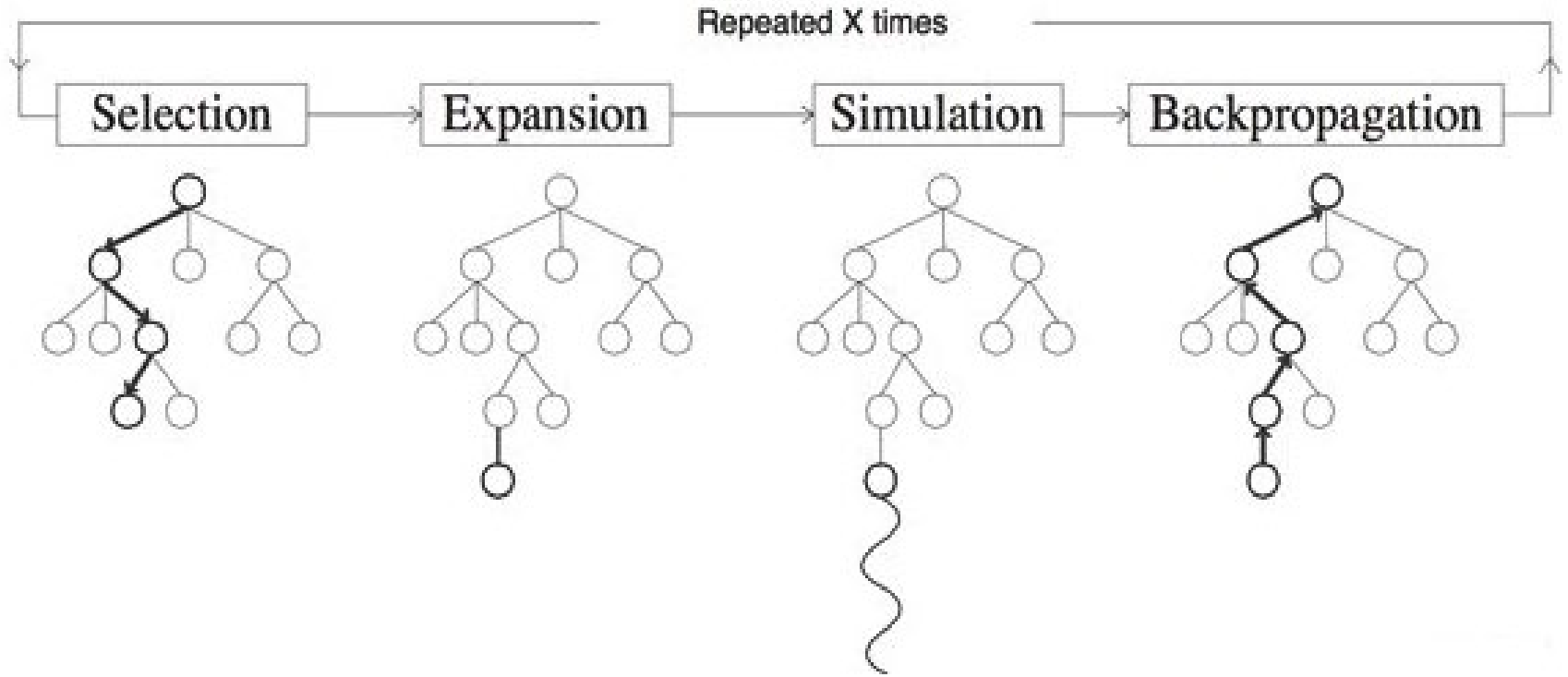
Deterministic policy: UCB1.

Initialization: Play each machine once.

Loop: Play machine j that maximizes $\bar{x}_j + \sqrt{\frac{2 \ln n}{n_j}}$, where $\bar{x}_j$ is the average reward obtained from machine j, n_j is the number of times machine j has been explored so far, and n is the overall number of plays done so far.

蒙特卡洛树搜索(Monte-Carlo Tree Search, MCTS)

- 将上限置信区间算法UCB应用于游戏树的搜索方法
  - 包括了四个步骤：选择(selection)，扩展(expansion)，模拟(simulation)，反向传播(Back-Propagation)

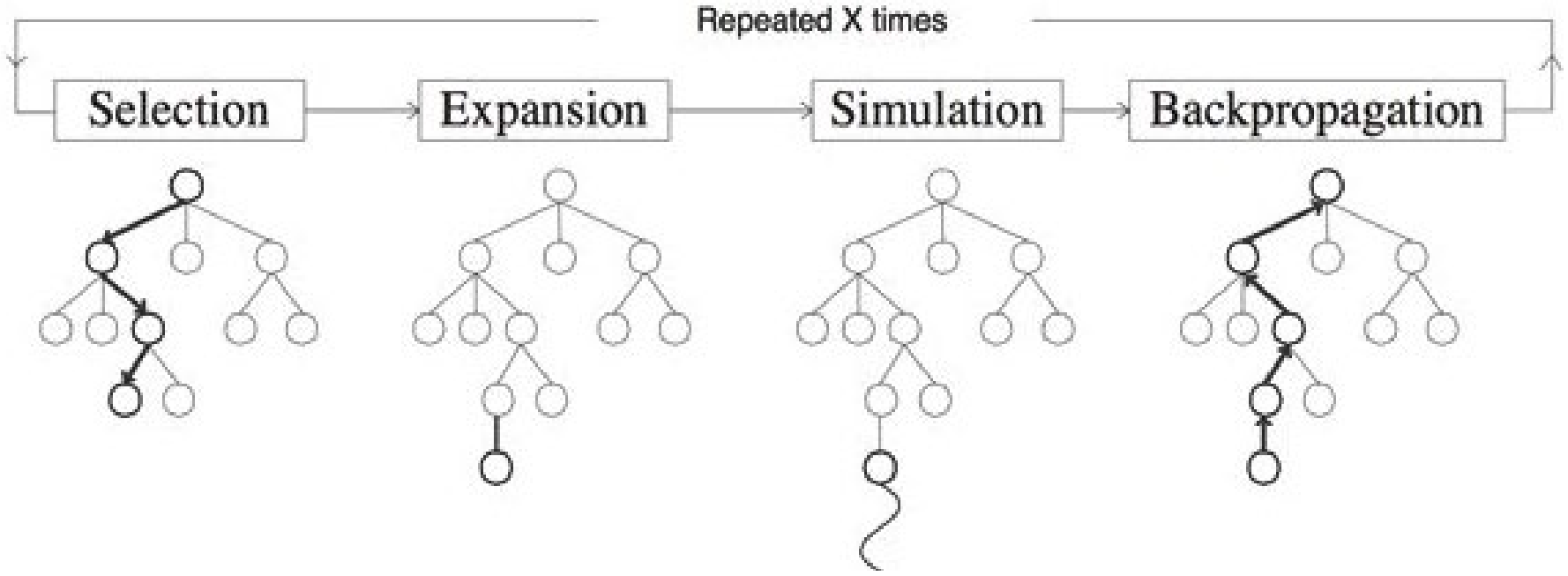


L. Kocsis and C. Szepesvari, Bandit based Monte-Carlo Planning, ***ECML***, 2006:282–293

蒙特卡洛树搜索

- **选择： 从根节点 R 开始，递归选择子节点，直至到达叶节点或到达具有还未被扩展过的子节点的节点L。**
  - 具体来说，通常用UCB1 (Upper Confidence Bound，上限置信区间)选择最具“潜力”的后续节点

$$UCB = \bar{X}_j + \sqrt{\frac{2\ln n}{n_j}}$$

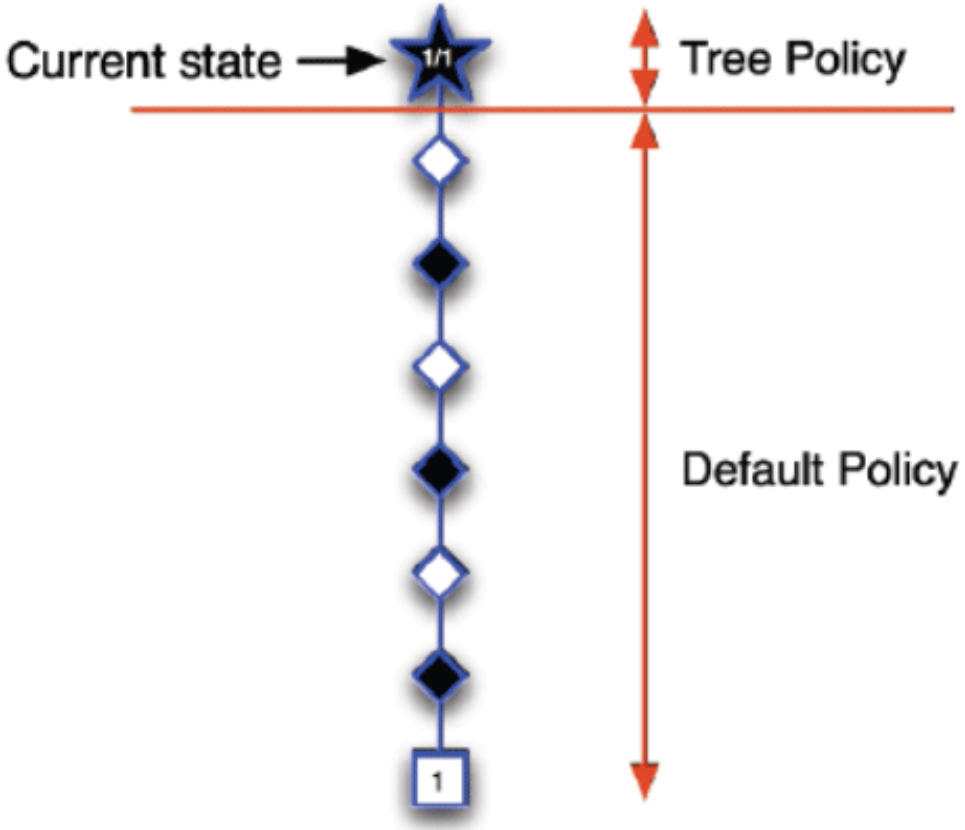


蒙特卡洛树搜索

- **扩展：**
  - 如果L不是一个终止节点，则随机创建其后的一个未被访问节点，选择该节点作为后续子节点C。
- **模拟：**
  - 从节点C出发，对游戏进行模拟，直到博弈游戏结束。
- **反向传播**
  - 用模拟所得结果来回溯更新导致这个结果的每个节点中获胜次数和访问次数。

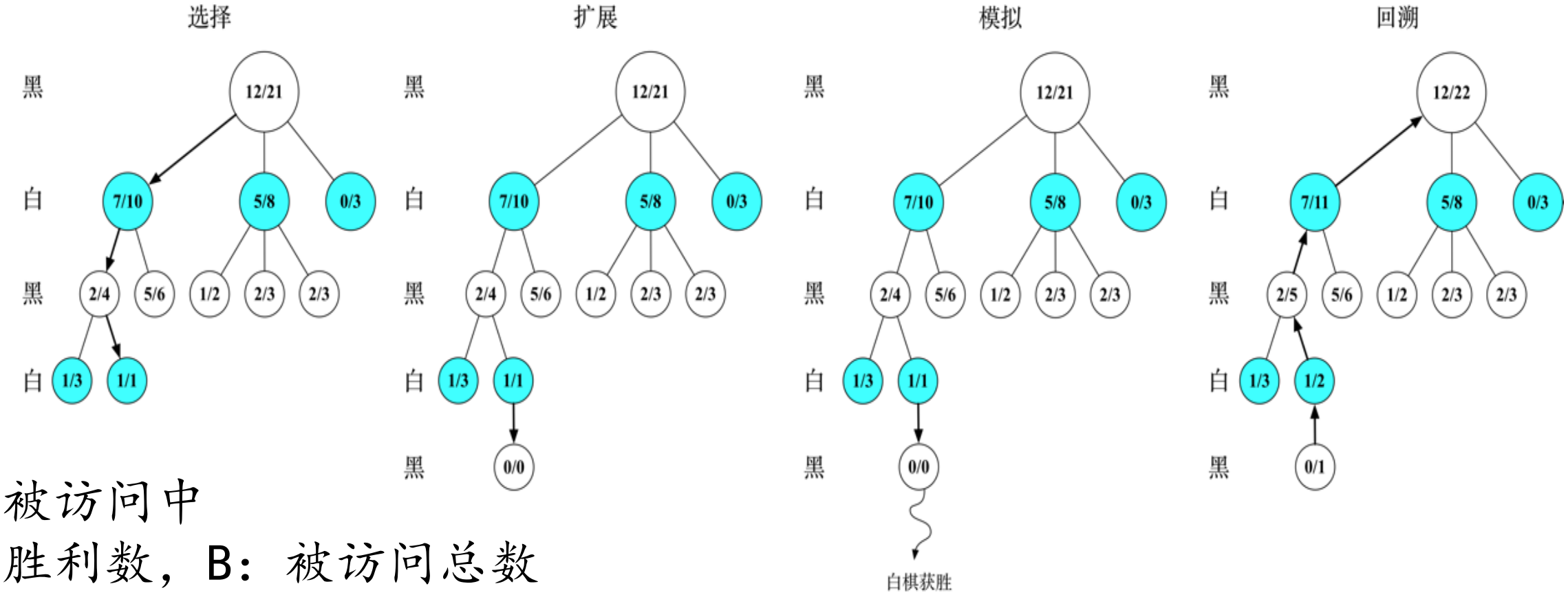
蒙特卡洛树搜索

- **两种策略学习机制：**
  - **搜索树策略**：从已有的搜索树中选择或创建一个叶子结点(即**选择**和**拓展**)。搜索树策略需要在利用和探索之间保持平衡。
  - **模拟策略**：从非叶子结点出发模拟游戏，得到游戏仿真结果。



蒙特卡洛树搜索：例子

- 以围棋为例，假设根节点是执黑棋方。
- 图中每一个节点都代表一个局面，每一个局面记录两个值A/B：



该局面被访问中

A:黑棋胜利数，B：被访问总数

蒙特卡洛树搜索：例子

- 假设由黑棋行棋，取一个UCB1值最大的节点作为后续节点：

左1：7/10对应

$$\frac{7}{10}+\sqrt{\frac{\log(21)}{10}}=1.252$$

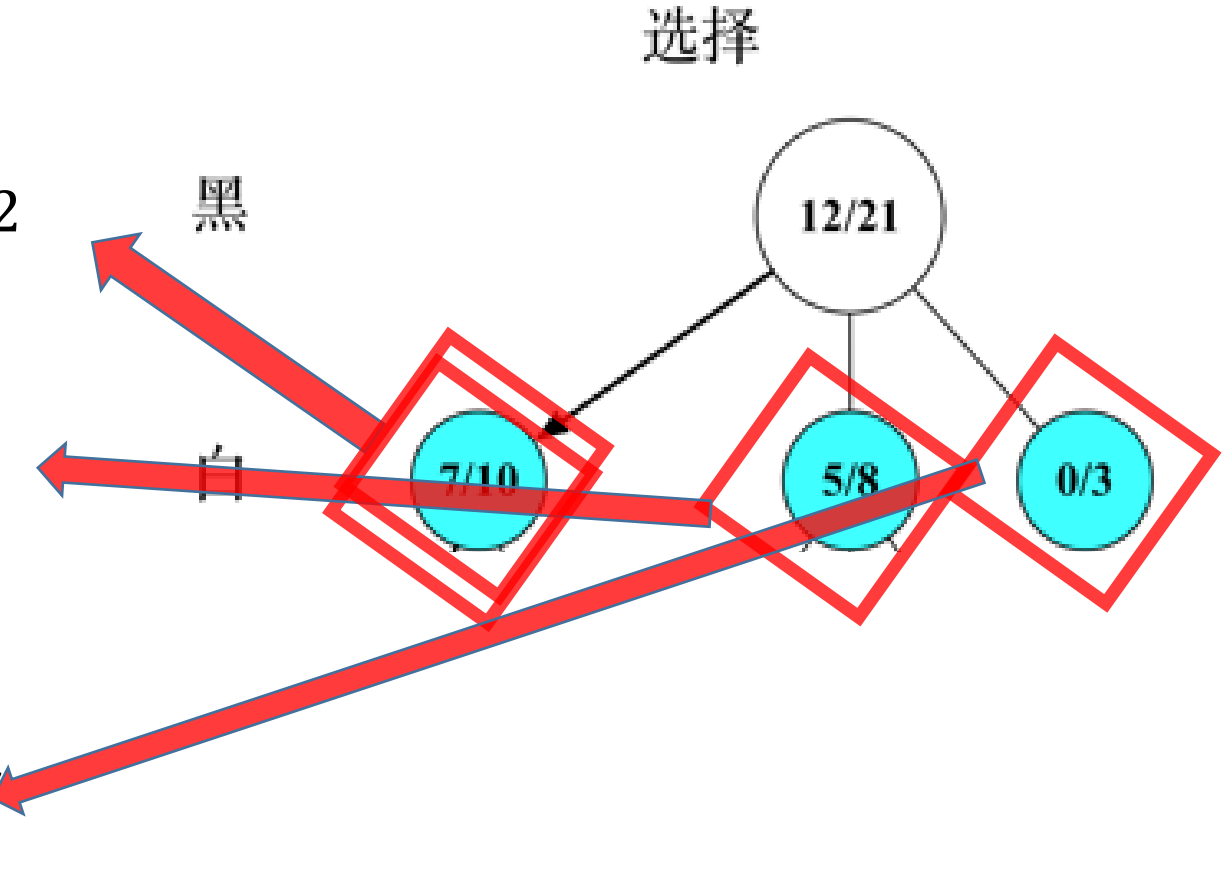
左2，5/8对应

$$\frac{5}{8}+\sqrt{\frac{\log(21)}{8}}=1.243$$

左3，0/3对应

$$\frac{0}{3}+\sqrt{\frac{\log(21)}{3}}=1.007$$

黑棋会选择局面7/10



蒙特卡洛树搜索：例子

- 在节点7/10，由白棋行棋(第一项为1-A/B)，由UCB1公式可得

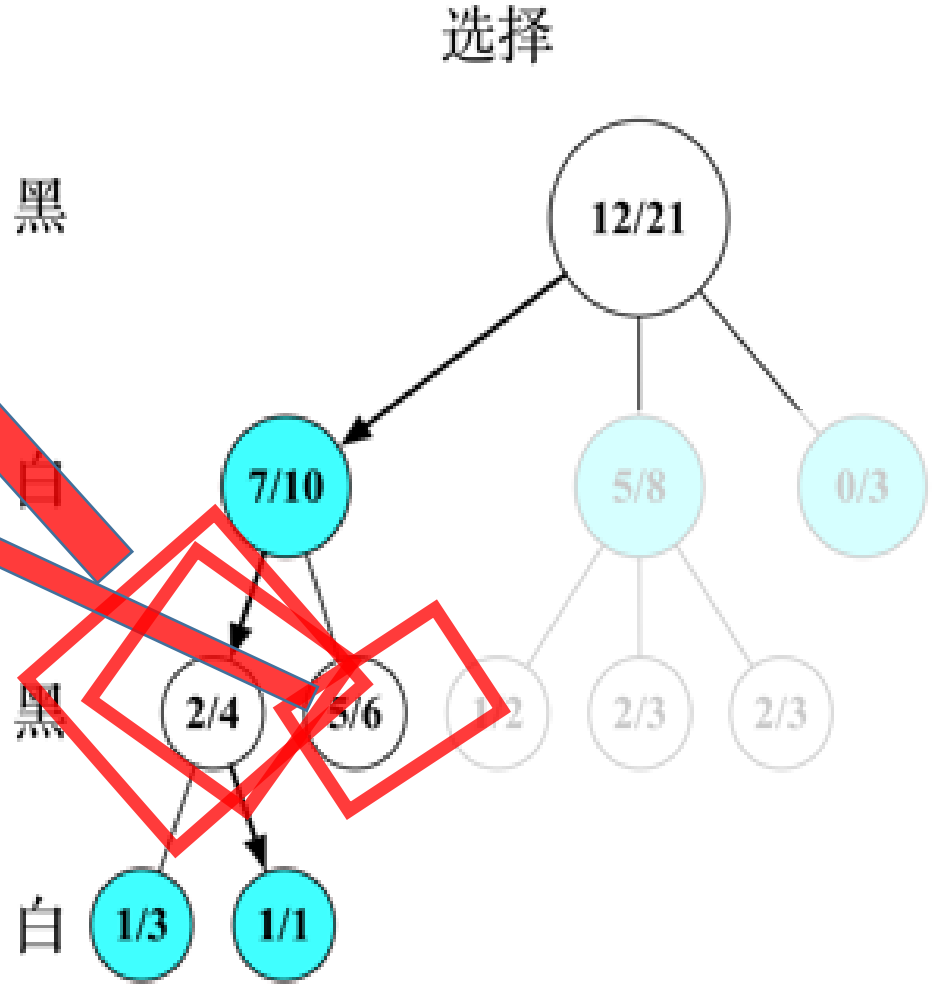
左1，2/4对应

$$\frac{2}{4}+\sqrt{\frac{\log(10)}{4}}=1.26$$

左2，5/6对应

$$\frac{1}{6}+\sqrt{\frac{\log(10)}{6}}=0.786$$

白棋会选择局面2/4



蒙特卡洛树搜索：例子

- 在节点2/4，黑棋评估下面的两个局面，由UCB1公式可得

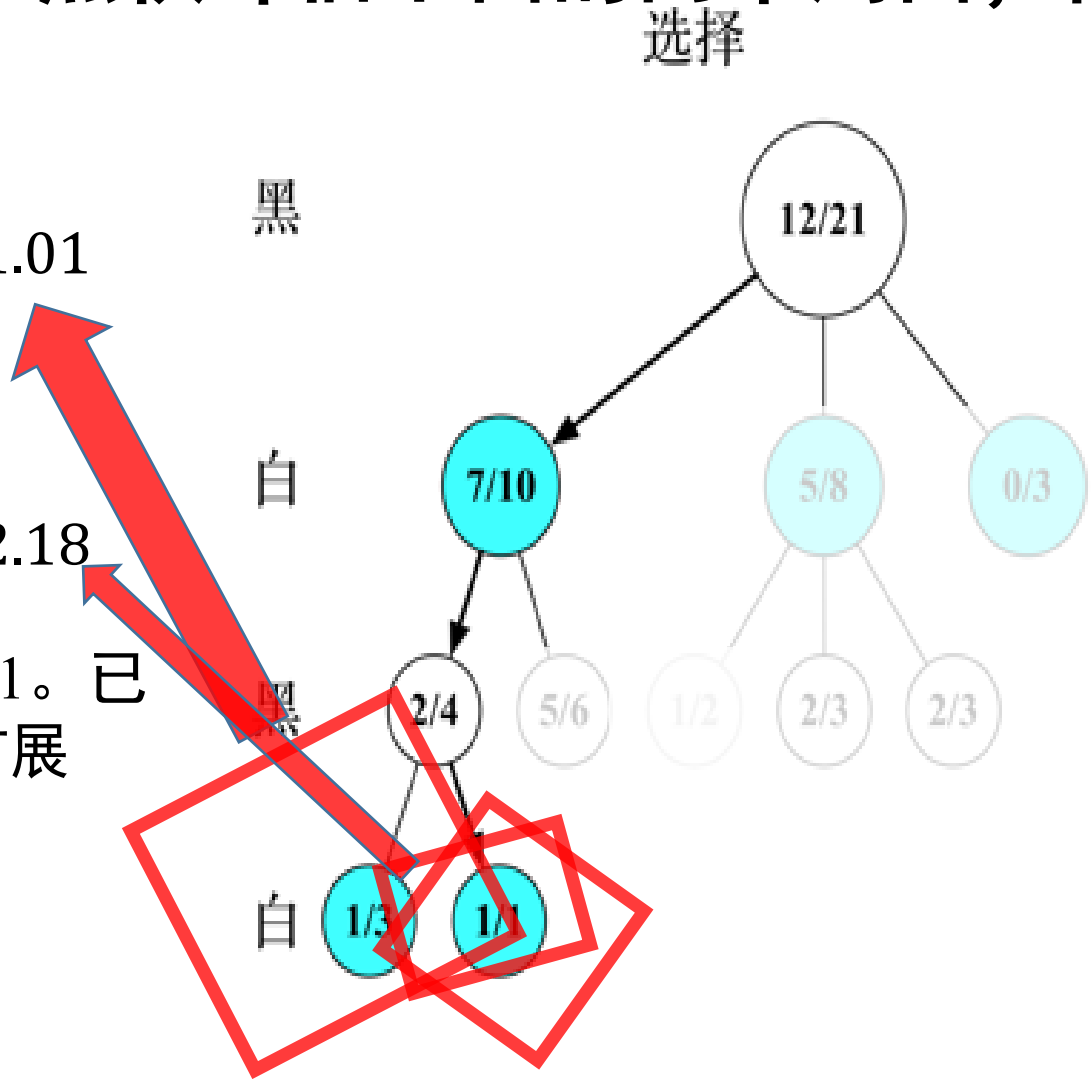
左1，1/3对应

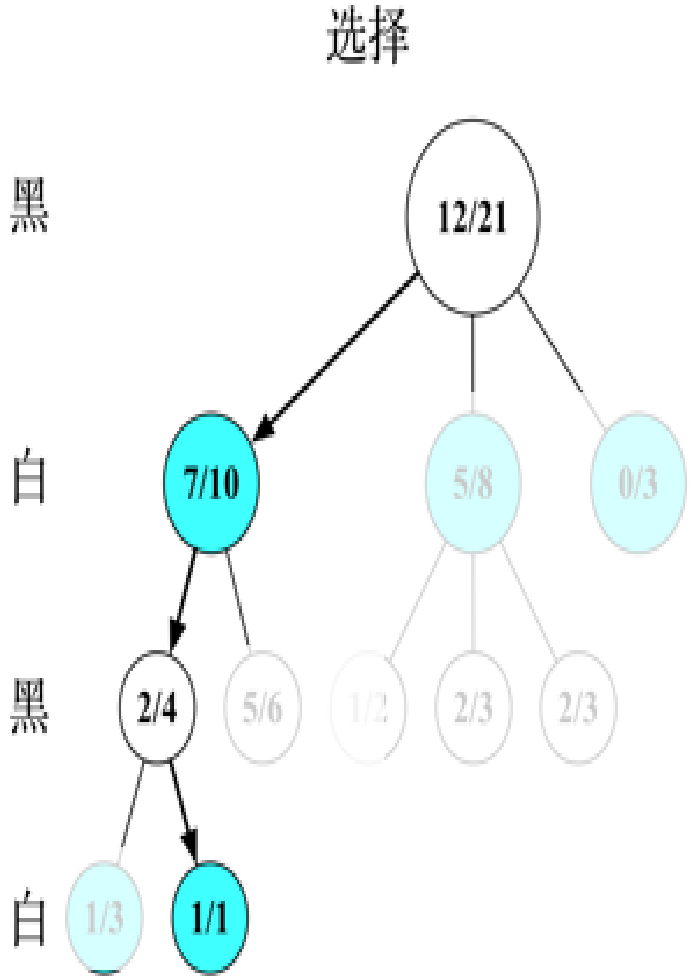
$$\frac{1}{3}+\sqrt{\frac{\log(4)}{3}}=1.01$$

左1，1/1对应

$$\frac{1}{1}+\sqrt{\frac{\log(4)}{1}}=2.18$$

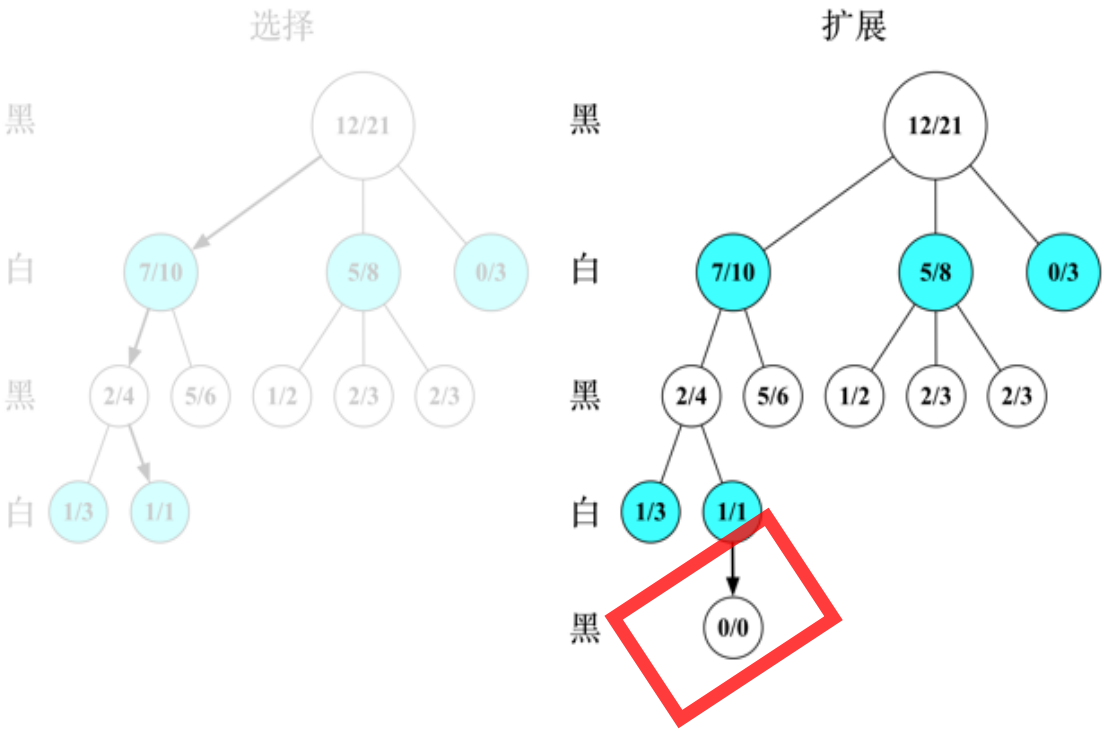
黑棋会选择局面1/1。已达叶结点，需要扩展





蒙特卡洛树搜索：例子

- 随机扩展一个新节点，初始化为0/0，在该节点下进行模拟



使用蒙特卡洛树搜索的原因

- 蒙特卡洛树搜索基于**采样**来得到结果、而非**穷尽式枚举** (虽然在枚举过程中也可剪掉若干不影响结果的分支)。

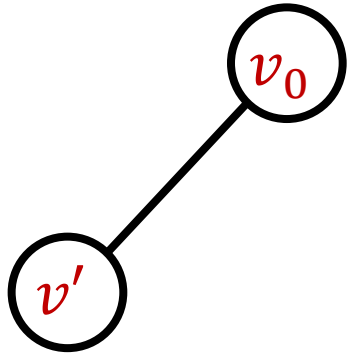
蒙特卡洛树搜索算法

S	状态集
$A(s)$	在状态s能够采取的有效行动的集合
$s(v)$	节点v所代表的状态
$a(v)$	所采取的行动导致到达节点v
$f:\ S \times A \rightarrow S$	状态转移函数
$N(v)$	节点v被访问的次数
$Q(V)$	节点v所获得的奖赏值
$\Delta(v,p)$	玩家p选择节点v所得到的奖赏值

蒙特卡洛树搜索算法(UCT)

S	状态集
$A(s)$	在状态s能够采取的有效行动的集合
$s(v)$	节点v所代表的状态
$a(v)$	所采取的行动导致到达节点v
$f : S \times A \to S$	状态转移函数
$N(v)$	节点v被访问的次数
$Q(V)$	节点v所获得的奖赏值
$\Delta(v,p)$	玩家p选择节点v所得到的奖赏值

function UCTSEARCH(s_0)
create root node v_0 with state s_0
while within computational budget **do**
$v_l \leftarrow$ TREEPOLICY(v_0)
$\Delta \leftarrow$ DEFAULTPOLICY($s(v_l)$)
BACKUP(v_l, Δ)
return a(BESTCHILD($v_0, 0$))



function TREEPOLICY(v)
while v is nonterminal **do**
if v not fully expanded **then**
return EXPAND(v)
else
$v \leftarrow$ BESTCHILD(v, Cp)
return v

function DEFAULTPOLICY(s)
while s is non-terminal **do**
choose $a \in A(s)$ uniformly at random
$s \leftarrow f(s, a)$
return reward for state s

function BESTCHILD(v, c)
return $\underset{v' \in \text{children of } v}{\arg\max} \frac{Q(v')}{N(v')} + c\sqrt{\frac{2 \ln N(v)}{N(v')}}$

function BACKUP(v, Δ)
while v is not null **do**
$N(v) \leftarrow N(v) + 1$
$Q(v) \leftarrow Q(v) + \Delta(v, p)$
$v \leftarrow$ parent of v

function EXPAND(v)
choose $a \in$ untried actions from $A(s(v))$
add a new child v' to v
with $s(v') = f(s(v), a)$
and $a(v') = a$
return v'

蒙特卡洛树搜索算法(UCT)

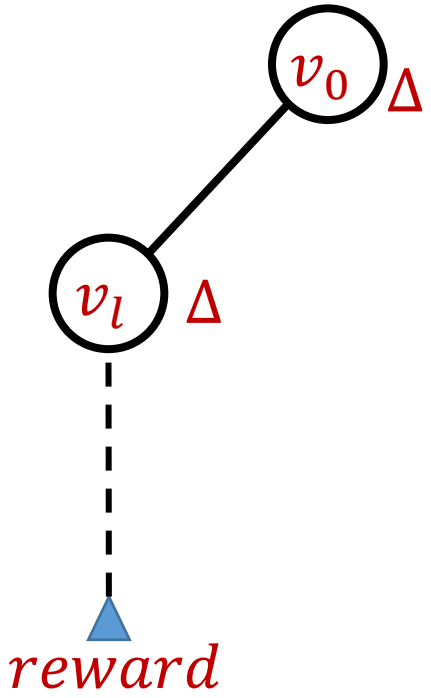
S	状态集
$A(s)$	在状态s能够采取的有效行动的集合
$s(v)$	节点v所代表的状态
$a(v)$	所采取的行动导致到达节点v
$f : S \times A \to S$	状态转移函数
$N(v)$	节点v被访问的次数
$Q(V)$	节点v所获得的奖赏值
$\Delta(v,p)$	玩家p选择节点v所得到的奖赏值

function UCTSEARCH(s_0)
  create root node v_0 with state s_0
  while within computational budget **do**
    $v_l \leftarrow$ TREEPOLICY(v_0)
    $\Delta \leftarrow$ DEFAULTPOLICY($s(v_l)$)
    BACKUP(v_l, Δ)
  return a(BESTCHILD($v_0, 0$))

function TREEPOLICY(v)
  while v is nonterminal **do**
    if v not fully expanded **then**
      return EXPAND(v)
    else
      $v \leftarrow$ BESTCHILD(v, Cp)
  return v

function DEFAULTPOLICY(s)
  while s is non-terminal **do**
    choose $a \in A(s)$ uniformly at random
    $s \leftarrow f(s, a)$
  return reward for state s

function BESTCHILD(v, c)
  return $\underset{v' \in \text{children of } v}{\arg\max} \frac{Q(v')}{N(v')} + c\sqrt{\frac{2 \ln N(v)}{N(v')}}$



function BACKUP(v, Δ)
  while v is not null **do**
    $N(v) \leftarrow N(v) + 1$
    $Q(v) \leftarrow Q(v) + \Delta(v, p)$
    $v \leftarrow$ parent of v

function EXPAND(v)
  choose $a \in$ untried actions from $A(s(v))$
  add a new child v' to v
    with $s(v') = f(s(v), a)$
    and $a(v') = a$
  return v'

蒙特卡洛树搜索算法(UCT)

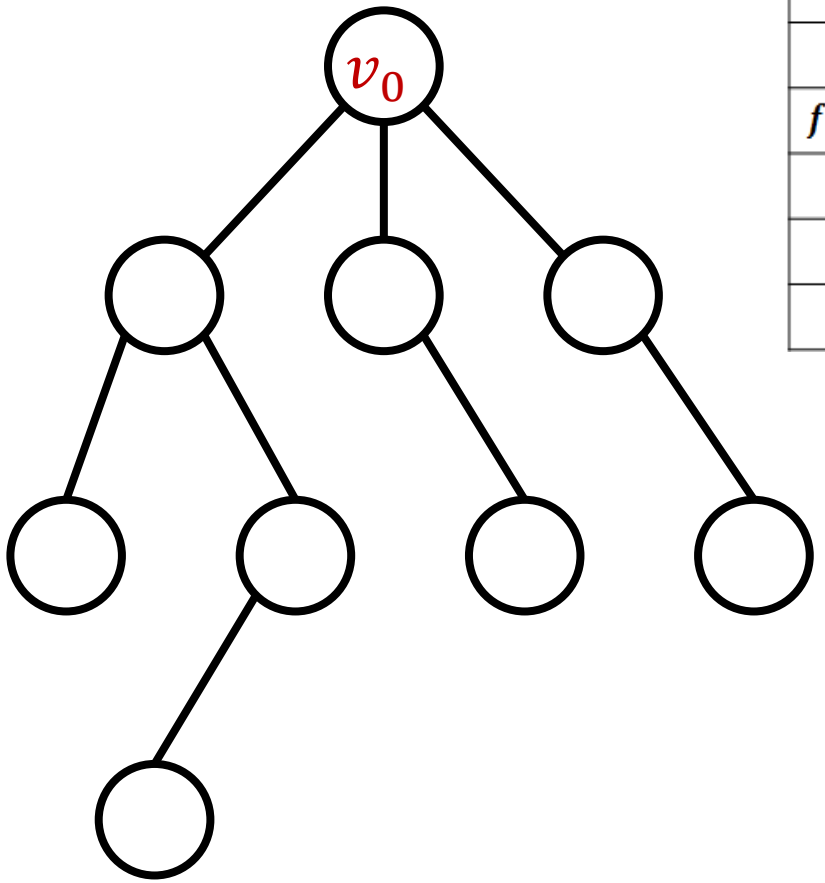
S	状态集
$A(s)$	在状态s能够采取的有效行动的集合
$s(v)$	节点v所代表的状态
$a(v)$	所采取的行动导致到达节点v
$f : S \times A \to S$	状态转移函数
$N(v)$	节点v被访问的次数
$Q(V)$	节点v所获得的奖赏值
$\Delta(v,p)$	玩家p选择节点v所得到的奖赏值

function UCTSEARCH(s_0)
  create root node v_0 with state s_0
  while within computational budget **do**
    $v_l \leftarrow$ TREEPOLICY(v_0)
    $\Delta \leftarrow$ DEFAULTPOLICY($s(v_l)$)
    BACKUP(v_l, Δ)
  return a(BESTCHILD($v_0, 0$))

function TREEPOLICY(v)
  while v is nonterminal **do**
    if v not fully expanded **then**
      return EXPAND(v)
    else
      $v \leftarrow$ BESTCHILD(v, Cp)
  return v

function DEFAULTPOLICY(s)
  while s is non-terminal **do**
    choose $a \in A(s)$ uniformly at random
    $s \leftarrow f(s, a)$
  return reward for state s

function BESTCHILD(v, c)
  return $\underset{v' \in \text{children of } v}{\arg\max} \frac{Q(v')}{N(v')} + c\sqrt{\frac{2\ln N(v)}{N(v')}}$



function BACKUP(v, Δ)
  while v is not null **do**
    $N(v) \leftarrow N(v) + 1$
    $Q(v) \leftarrow Q(v) + \Delta(v, p)$
    $v \leftarrow$ parent of v

function EXPAND(v)
  choose $a \in$ untried actions from $A(s(v))$
  add a new child v' to v
    with $s(v') = f(s(v), a)$
    and $a(v') = a$
  return v'

蒙特卡洛树搜索算法(UCT)

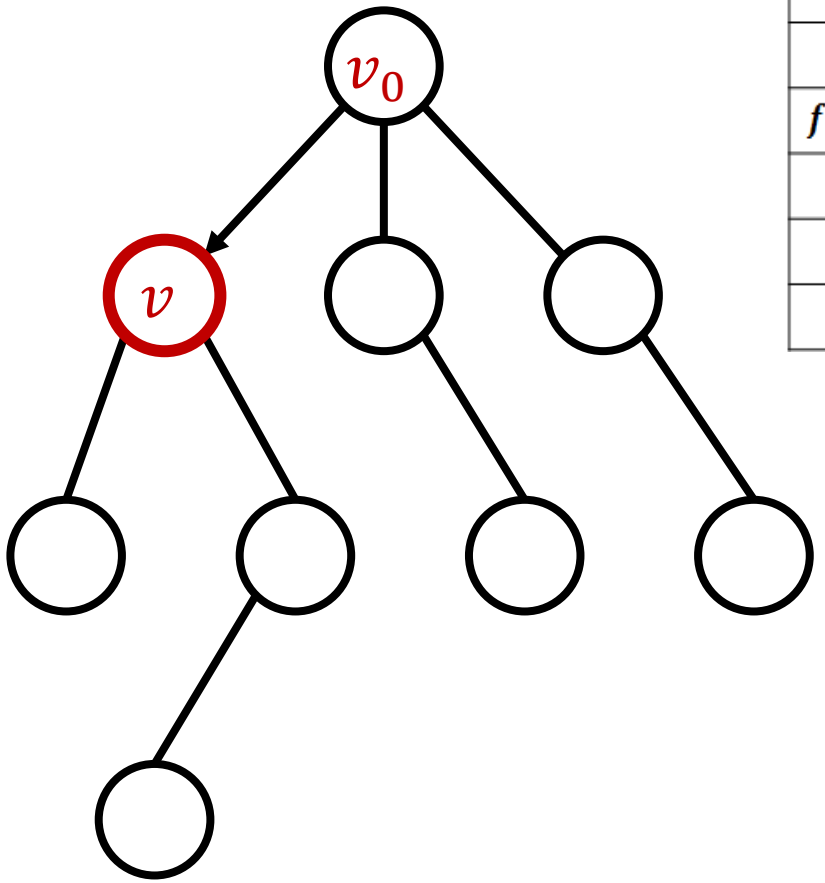
S	状态集
$A(s)$	在状态s能够采取的有效行动的集合
$s(v)$	节点v所代表的状态
$a(v)$	所采取的行动导致到达节点v
$f : S \times A \to S$	状态转移函数
$N(v)$	节点v被访问的次数
$Q(V)$	节点v所获得的奖赏值
$\Delta(v,p)$	玩家p选择节点v所得到的奖赏值

function UCTSEARCH(s_0)
  create root node v_0 with state s_0
  while within computational budget **do**
    $v_l \leftarrow$ TREEPOLICY(v_0)
    $\Delta \leftarrow$ DEFAULTPOLICY($s(v_l)$)
    BACKUP(v_l, Δ)
  return a(BESTCHILD($v_0, 0$))

function TREEPOLICY(v)
  while v is nonterminal **do**
    if v not fully expanded **then**
      return EXPAND(v)
    else
      $v \leftarrow$ BESTCHILD(v, Cp)
  return v

function DEFAULTPOLICY(s)
  while s is non-terminal **do**
    choose $a \in A(s)$ uniformly at random
    $s \leftarrow f(s, a)$
  return reward for state s

function BESTCHILD(v, c)
  return $\underset{v' \in \text{children of } v}{\arg\max} \frac{Q(v')}{N(v')} + c\sqrt{\frac{2 \ln N(v)}{N(v')}}$



function BACKUP(v, Δ)
  while v is not null **do**
    $N(v) \leftarrow N(v) + 1$
    $Q(v) \leftarrow Q(v) + \Delta(v, p)$
    $v \leftarrow$ parent of v

function EXPAND(v)
  choose $a \in$ untried actions from $A(s(v))$
  add a new child v' to v
    with $s(v') = f(s(v), a)$
    and $a(v') = a$
  return v'

蒙特卡洛树搜索算法(UCT)

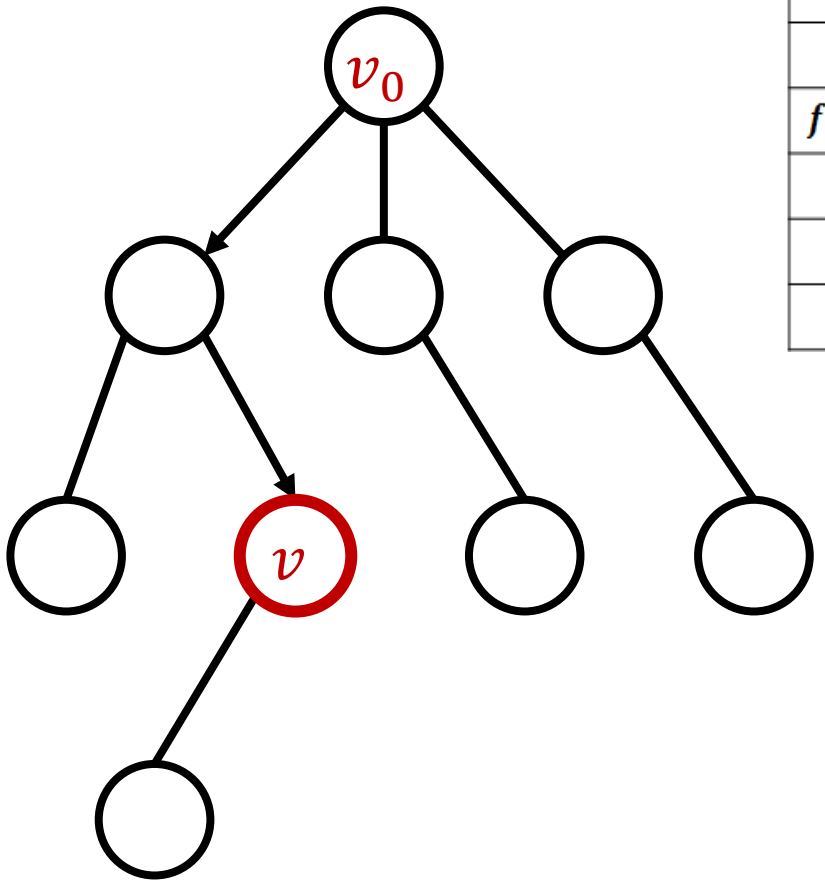
S	状态集
$A(s)$	在状态s能够采取的有效行动的集合
$s(v)$	节点v所代表的状态
$a(v)$	所采取的行动导致到达节点v
$f : S \times A \to S$	状态转移函数
$N(v)$	节点v被访问的次数
$Q(V)$	节点v所获得的奖赏值
$\Delta(v,p)$	玩家p选择节点v所得到的奖赏值

function UCTSEARCH(s_0)
  create root node v_0 with state s_0
  while within computational budget **do**
    $v_l \leftarrow$ TREEPOLICY(v_0)
    $\Delta \leftarrow$ DEFAULTPOLICY($s(v_l)$)
    BACKUP(v_l, Δ)
  return a(BESTCHILD($v_0, 0$))

function TREEPOLICY(v)
  while v is nonterminal **do**
    if v not fully expanded **then**
      return EXPAND(v)
    else
      $v \leftarrow$ BESTCHILD(v, Cp)
  return v

function DEFAULTPOLICY(s)
  while s is non-terminal **do**
    choose $a \in A(s)$ uniformly at random
    $s \leftarrow f(s, a)$
  return reward for state s

function BESTCHILD(v, c)
  return $\underset{v' \in \text{children of } v}{\arg\max} \frac{Q(v')}{N(v')} + c\sqrt{\frac{2 \ln N(v)}{N(v')}}$



function BACKUP(v, Δ)
  while v is not null **do**
    $N(v) \leftarrow N(v) + 1$
    $Q(v) \leftarrow Q(v) + \Delta(v, p)$
    $v \leftarrow$ parent of v

function EXPAND(v)
  choose $a \in$ untried actions from $A(s(v))$
  add a new child v' to v
    with $s(v') = f(s(v), a)$
    and $a(v') = a$
  return v'

蒙特卡洛树搜索算法(UCT)

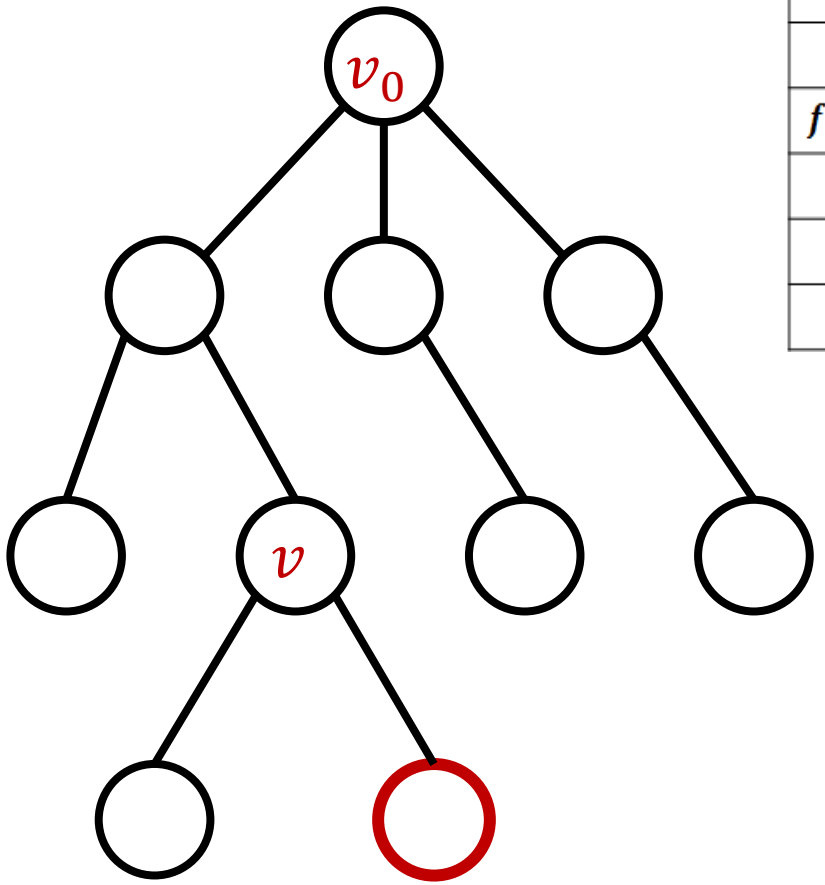
S	状态集
$A(s)$	在状态s能够采取的有效行动的集合
$s(v)$	节点v所代表的状态
$a(v)$	所采取的行动导致到达节点v
$f : S \times A \to S$	状态转移函数
$N(v)$	节点v被访问的次数
$Q(V)$	节点v所获得的奖赏值
$\Delta(v,p)$	玩家p选择节点v所得到的奖赏值

function UCTSEARCH(s_0)
  create root node v_0 with state s_0
  while within computational budget **do**
    $v_l \leftarrow$ TREEPOLICY(v_0)
    $\Delta \leftarrow$ DEFAULTPOLICY($s(v_l)$)
    BACKUP(v_l, Δ)
  return a(BESTCHILD($v_0, 0$))

function TREEPOLICY(v)
  while v is nonterminal **do**
    if v not fully expanded **then**
      return EXPAND(v)
    else
      $v \leftarrow$ BESTCHILD(v, Cp)
  return v

function DEFAULTPOLICY(s)
  while s is non-terminal **do**
    choose $a \in A(s)$ uniformly at random
    $s \leftarrow f(s, a)$
  return reward for state s

function BESTCHILD(v, c)
  return $\underset{v' \in \text{children of } v}{\arg\max} \frac{Q(v')}{N(v')} + c\sqrt{\frac{2\ln N(v)}{N(v')}}$



function BACKUP(v, Δ)
  while v is not null **do**
    $N(v) \leftarrow N(v) + 1$
    $Q(v) \leftarrow Q(v) + \Delta(v, p)$
    $v \leftarrow$ parent of v

function EXPAND(v)
  choose $a \in$ untried actions from $A(s(v))$
  add a new child v' to v
    with $s(v') = f(s(v), a)$
    and $a(v') = a$
  return v'

蒙特卡洛树搜索算法(UCT)

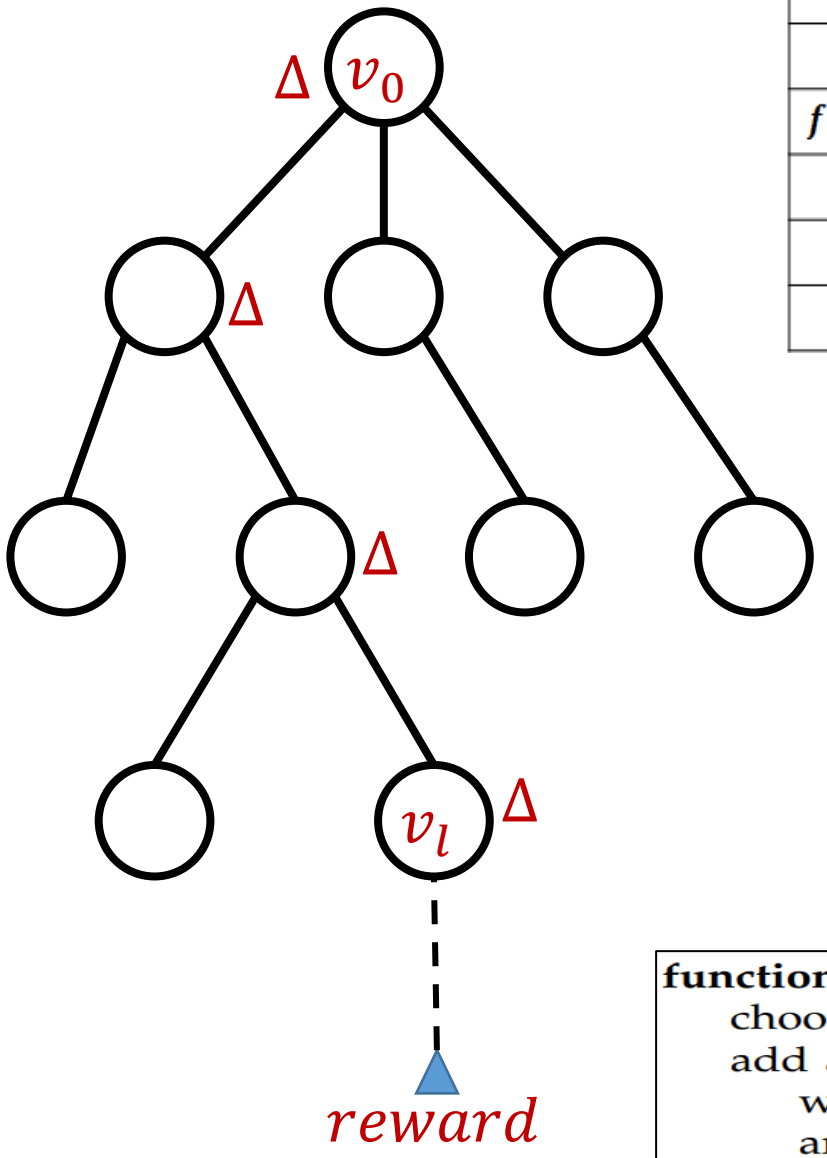
S	状态集
$A(s)$	在状态s能够采取的有效行动的集合
$s(v)$	节点v所代表的状态
$a(v)$	所采取的行动导致到达节点v
$f : S \times A \to S$	状态转移函数
$N(v)$	节点v被访问的次数
$Q(V)$	节点v所获得的奖赏值
$\Delta(v,p)$	玩家p选择节点v所得到的奖赏值

function UCTSEARCH(s_0)
  create root node v_0 with state s_0
  while within computational budget **do**
    $v_l \leftarrow$ TREEPOLICY(v_0)
    $\Delta \leftarrow$ DEFAULTPOLICY($s(v_l)$)
    BACKUP(v_l, Δ)
  return a(BESTCHILD($v_0, 0$))

function TREEPOLICY(v)
  while v is nonterminal **do**
    if v not fully expanded **then**
      return EXPAND(v)
    else
      $v \leftarrow$ BESTCHILD(v, Cp)
  return v

function DEFAULTPOLICY(s)
  while s is non-terminal **do**
    choose $a \in A(s)$ uniformly at random
    $s \leftarrow f(s, a)$
  return reward for state s

function BESTCHILD(v, c)
  return $\underset{v' \in \text{children of } v}{\arg\max} \frac{Q(v')}{N(v')} + c\sqrt{\frac{2 \ln N(v)}{N(v')}}$



function BACKUP(v, Δ)
  while v is not null **do**
    $N(v) \leftarrow N(v) + 1$
    $Q(v) \leftarrow Q(v) + \Delta(v, p)$
    $v \leftarrow$ parent of v

function EXPAND(v)
  choose $a \in$ untried actions from $A(s(v))$
  add a new child v' to v
    with $s(v') = f(s(v), a)$
    and $a(v') = a$
  return v'

蒙特卡洛树搜索算法(UCT)

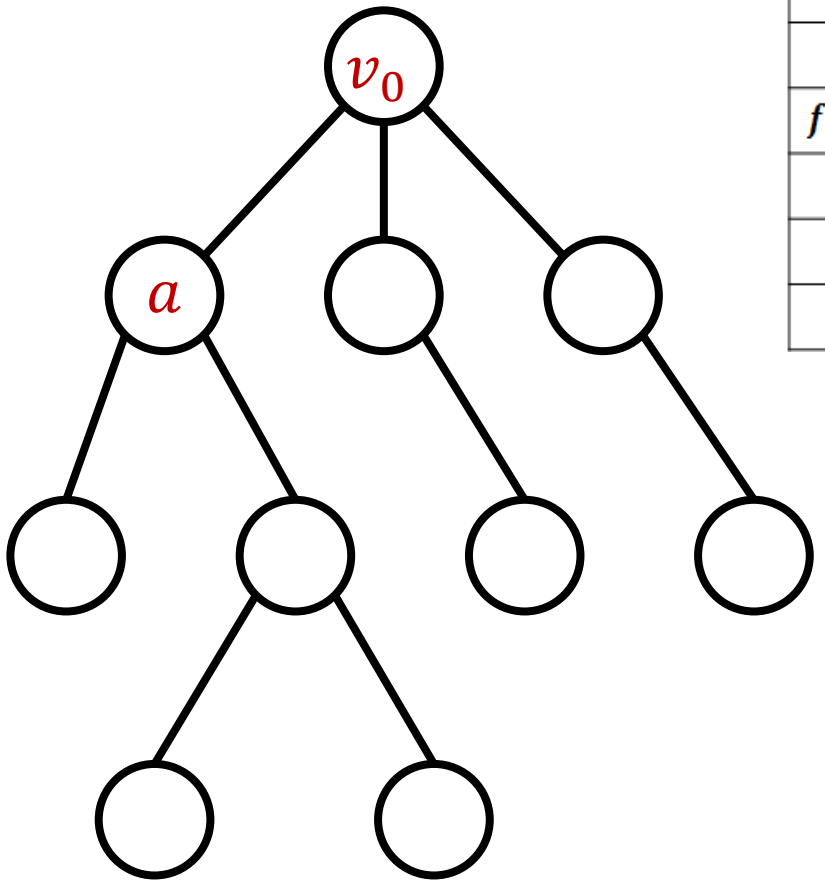
S	状态集
$A(s)$	在状态s能够采取的有效行动的集合
$s(v)$	节点v所代表的状态
$a(v)$	所采取的行动导致到达节点v
$f : S \times A \to S$	状态转移函数
$N(v)$	节点v被访问的次数
$Q(V)$	节点v所获得的奖赏值
$\Delta(v,p)$	玩家p选择节点v所得到的奖赏值

function UCTSEARCH(s_0)
  create root node v_0 with state s_0
  while within computational budget **do**
    $v_l \leftarrow$ TREEPOLICY(v_0)
    $\Delta \leftarrow$ DEFAULTPOLICY($s(v_l)$)
    BACKUP(v_l, Δ)
  return a(BESTCHILD($v_0, 0$))

function TREEPOLICY(v)
  while v is nonterminal **do**
    if v not fully expanded **then**
      return EXPAND(v)
    else
      $v \leftarrow$ BESTCHILD(v, Cp)
  return v

function DEFAULTPOLICY(s)
  while s is non-terminal **do**
    choose $a \in A(s)$ uniformly at random
    $s \leftarrow f(s, a)$
  return reward for state s

function BESTCHILD(v, c)
  return $\underset{v' \in \text{children of } v}{\arg\max} \frac{Q(v')}{N(v')} + c\sqrt{\frac{2 \ln N(v)}{N(v')}}$



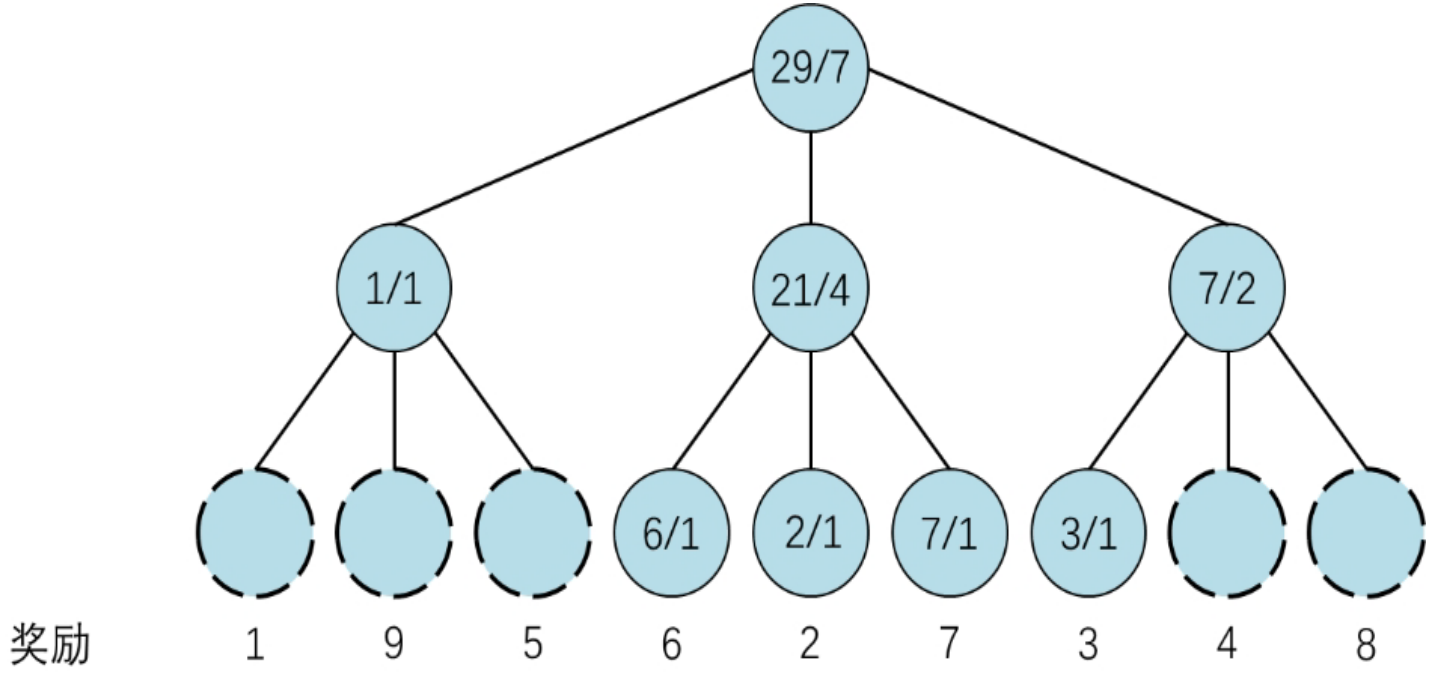
function BACKUP(v, Δ)
  while v is not null **do**
    $N(v) \leftarrow N(v) + 1$
    $Q(v) \leftarrow Q(v) + \Delta(v, p)$
    $v \leftarrow$ parent of v

function EXPAND(v)
  choose $a \in$ untried actions from $A(s(v))$
  add a new child v' to v
    with $s(v') = f(s(v), a)$
    and $a(v') = a$
  return v'

课上习题

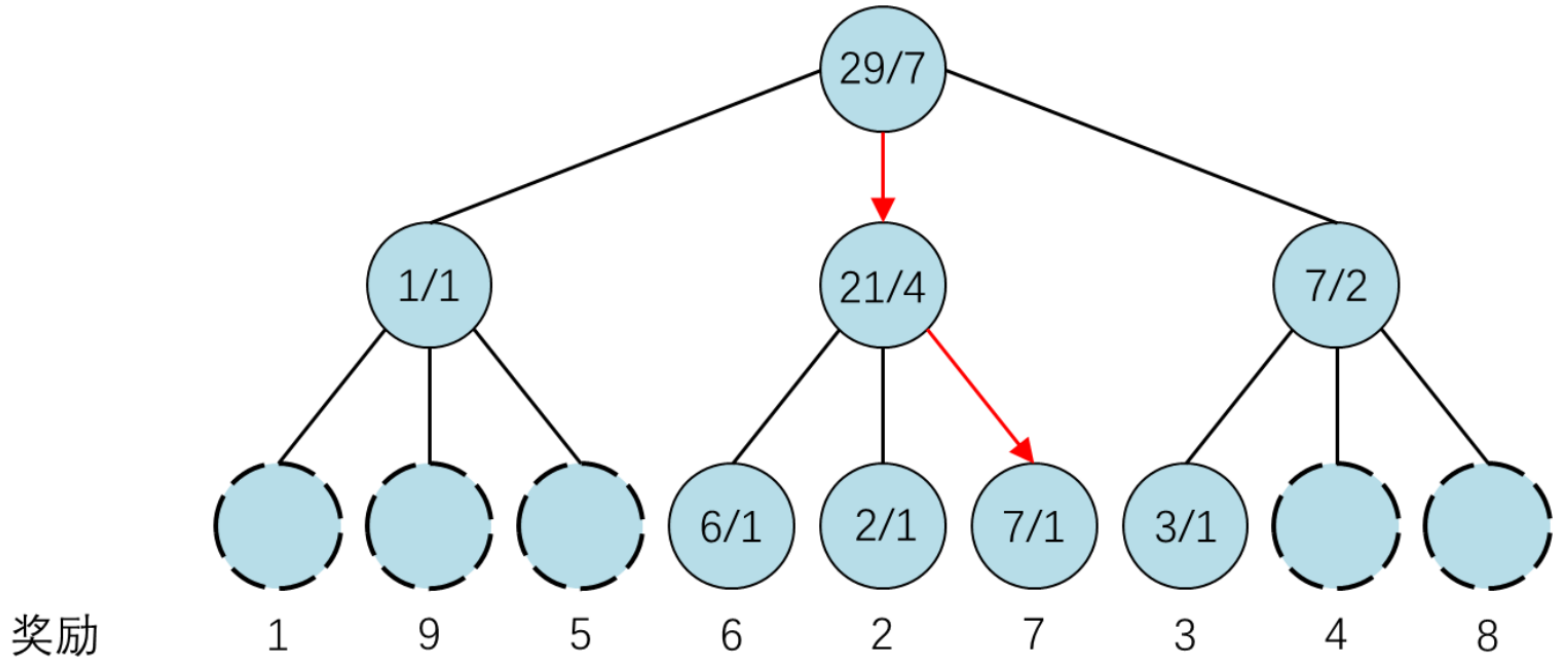
图 3 展示了一个蒙特卡洛树搜索的例子。其中每个叶子节点（终止节点）下标出了该节点对应的奖励。为了最大化取得的奖励，可利用蒙特卡洛树搜索求解奖励最大的路径。假设执行了若干步骤后，算法的状态如图 3 所示，节点内的数字分别表示“总奖励/访问次数”，虚线节点表示尚未扩展的节点。算法此时正要开始下一轮选择-扩展-模拟-反向传播的迭代

(1) 假设 UCB1 算法中的超参数$C = 1$，请计算并画出算法选择过程经过的路径。



课上习题

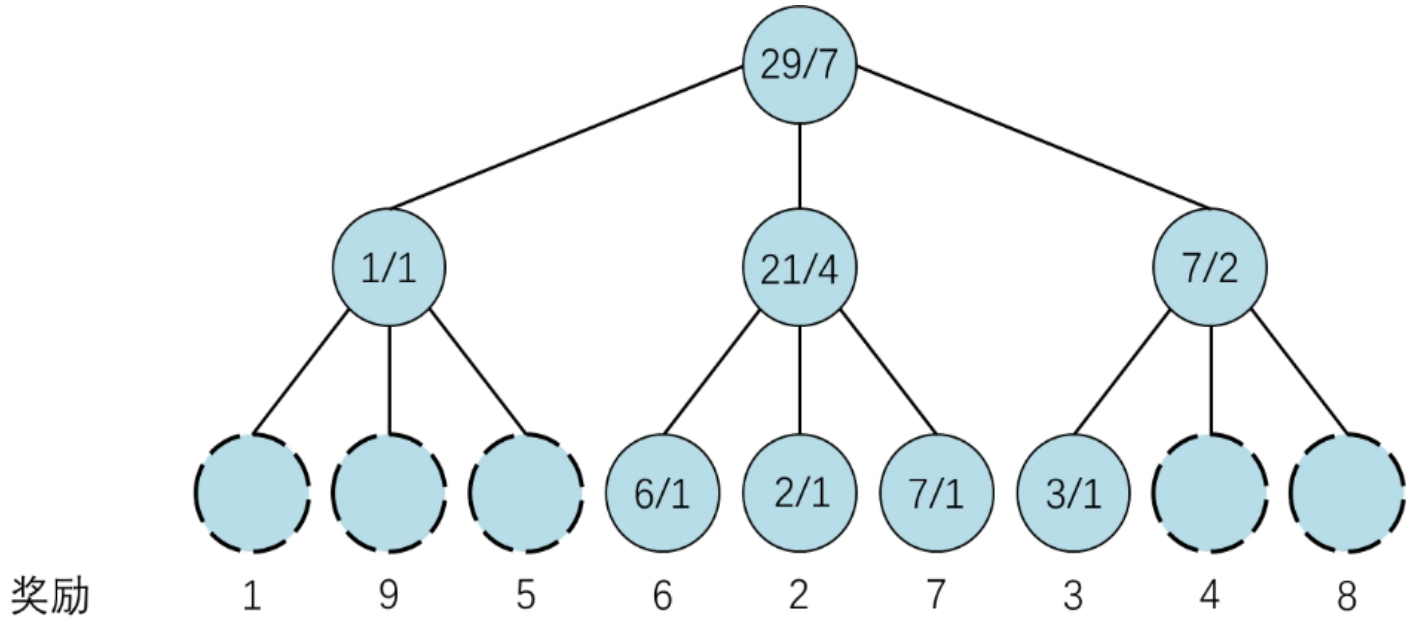
(1) 第一步三个节点的 UCB 值从左到右分别为$\frac{1}{1}+\sqrt{\frac{2\ln 7}{1}}=2.97$, $\frac{21}{4}+\sqrt{\frac{2\ln 7}{4}}=6.24$, $\frac{7}{2}+\sqrt{\frac{2\ln 7}{2}}=4.89$，因此第一步选择第二层中间的节点。第二步三个节点的 UCB 值从左到右分别为$\frac{6}{1}+\sqrt{\frac{2\ln 4}{1}}=7.67$，$\frac{2}{1}+\sqrt{\frac{2\ln 4}{1}}=3.67$，$\frac{7}{1}+\sqrt{\frac{2\ln 4}{1}}=8.67$，因此第二步选择奖励为 7 的节点。如下图所示。



课上习题

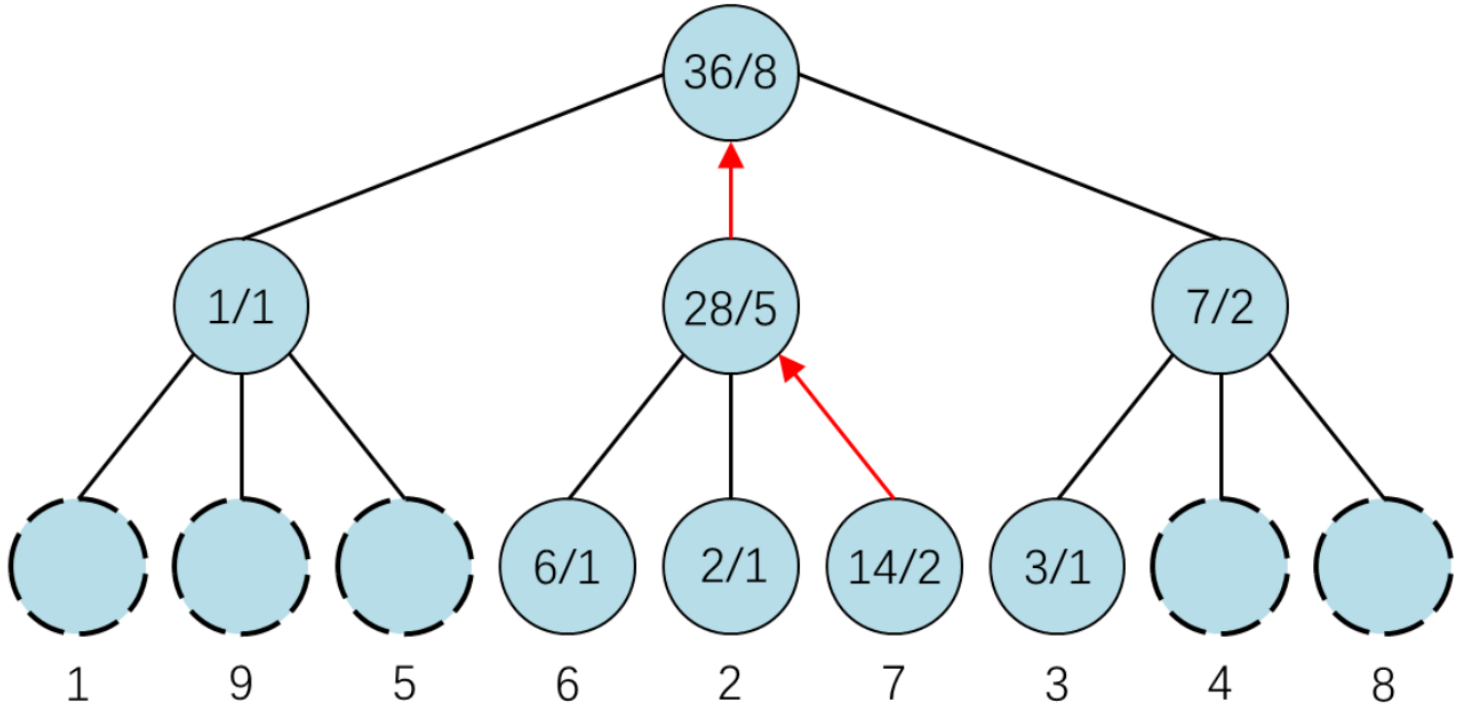
图 3 展示了一个蒙特卡洛树搜索的例子。其中每个叶子节点（终止节点）下标出了该节点对应的奖励。为了最大化取得的奖励，可利用蒙特卡洛树搜索求解奖励最大的路径。假设执行了若干步骤后，算法的状态如图 3 所示，节点内的数字分别表示“总奖励/访问次数”，虚线节点表示尚未扩展的节点。算法此时正要开始下一轮选择-扩展-模拟-反向传播的迭代

(2) 请继续执行扩展、模拟、反向传播步骤，并画出完成后的搜索树状态。（为了避免随机性，假设扩展总是扩展最左侧的未扩展节点，模拟总是选择最左侧的路径。）



课上习题

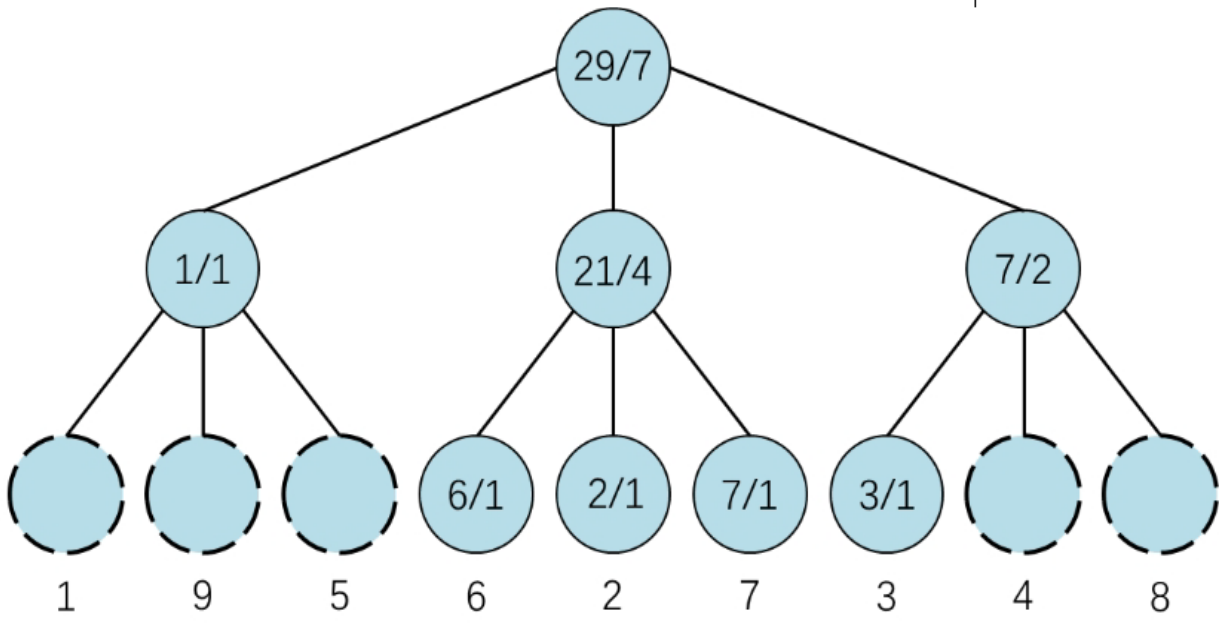
(2) 由于此时已经到达叶子节点，因此不需要进行扩展和模拟过程，反向传播后结果如下图所示



课上习题

图 3 展示了一个蒙特卡洛树搜索的例子。其中每个叶子节点（终止节点）下标出了该节点对应的奖励。为了最大化取得的奖励，可利用蒙特卡洛树搜索求解奖励最大的路径。假设执行了若干步骤后，算法的状态如图 3 所示，节点内的数字分别表示“总奖励/访问次数”，虚线节点表示尚未扩展的节点。算法此时正要开始下一轮选择-扩展-模拟-反向传播的迭代

(3) 尝试进行若干次迭代，请问此时算法是否能有效地找到奖励最大的叶子结点（奖励为 9），那么进行足够多次迭代以后又如何？如果希望提高算法的效率，应该做出怎样的调整？

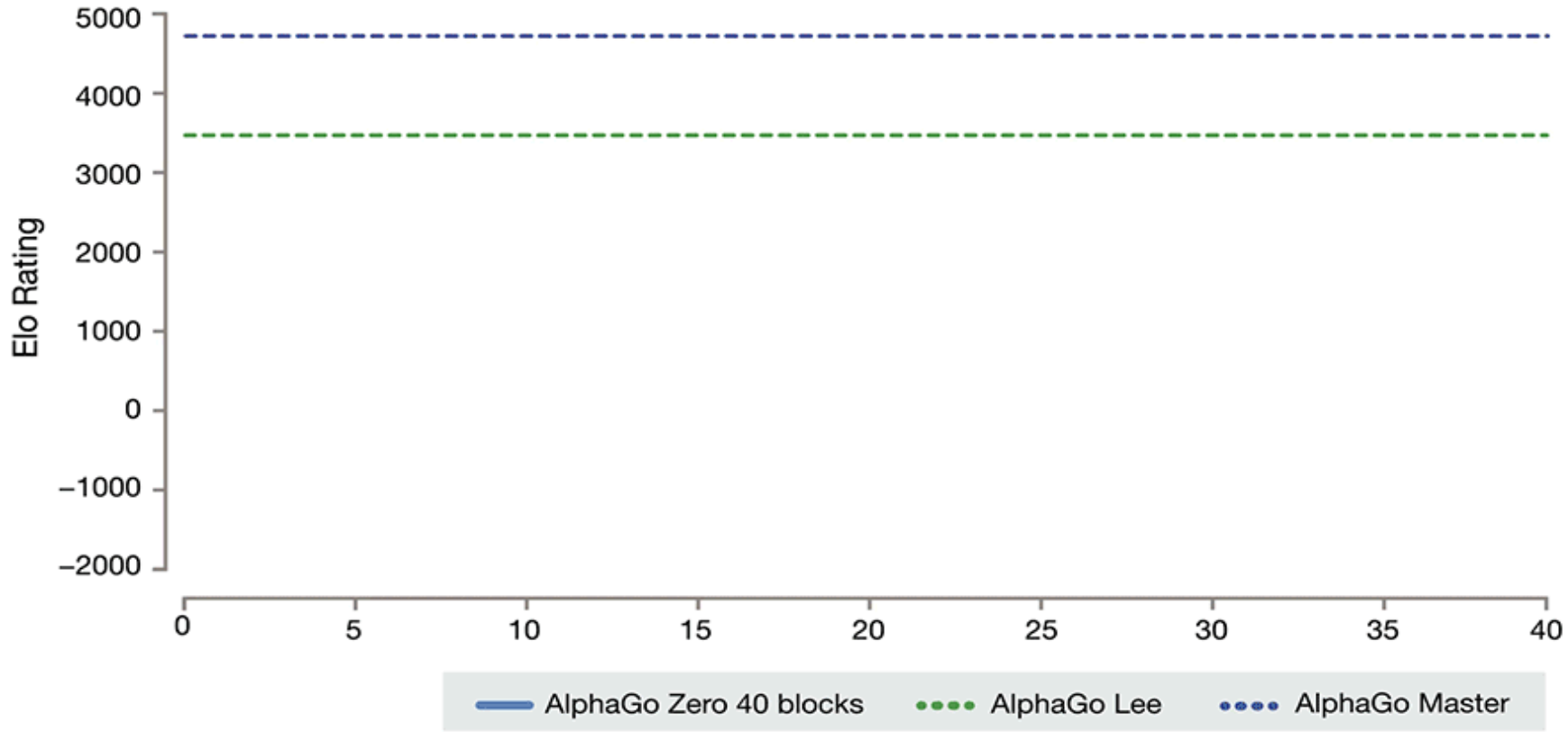


课上习题

(3) 算法在很长一段时间内都会选择奖励为 7 的节点，而不会探索奖励为 9 的节点。当实验次数足够多时，第二层左侧的节点的 UCB 值最终会超过第二层中间节点的 UCB 值，因此只要实验次数足够多，算法是有可能探索到奖励为 9 的节点的。如果希望提高算法的效率，可考虑加大探索的力度，即取一个更大的超参数C。不难验证，在原题中的状态下，取$C = 10$即可令算法选择第二层左侧的节点。

AlphaGo算法解读：AlphaGo Zero

- 经过40天训练后，Zero总计运行约2900万次自我对弈，得以击败AlphaGo Master，比分为89比11



Mastering the game of Go without human knowledge, *Nature*, volume 550, pages 354–359, 2017

拓展学习 (AlphaGo)

- 链接

阿尔法狗 | AlphaGo 万字解读 【下集】

5844播放 · 总弹幕数12　2021-03-20 13:20:01

谢谢!