中国地质大学 China University of Geosciences

第五章

对 抗 搜 索 和 博 弈

Adversarial Search and Game Playing

计算机学院 人工智能课程

赵 曼

目　录

目　录

Zhaoman, AI,
Department of Computer Science, CS, CUG

Games are Good Problems for AI 博弈是AI研究的好材料

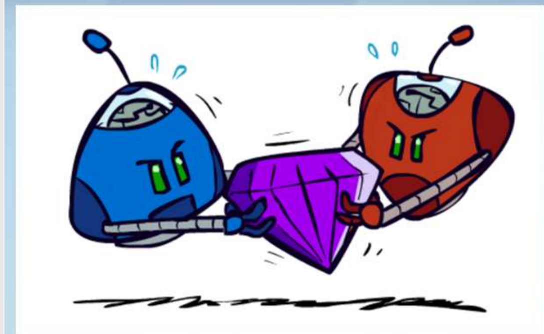
- Machines (players) need “human-like” intelligence.
  机器（玩家）需要“类人”的智能。
- Requiring to make decision within limited time.
  要求在有限的时间内进行决策。

Features of games 博弈的特征：

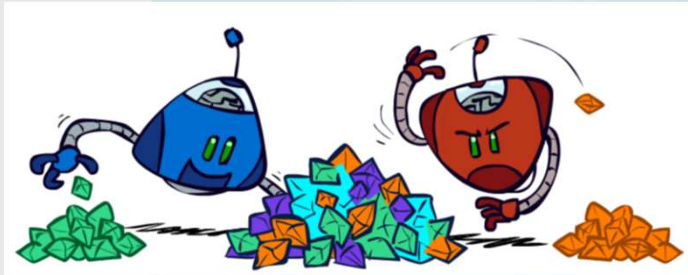
Two, or more players (agents)	■	两个、或多个玩家（智能体）
Turn-taking vs. simultaneous moves	■	轮流与同步行动
Perfect information vs. imperfect information	■	完全信息与不完全信息
Deterministic vs. stochastic	■	确定性与随机
competitive vs. Cooperative	■	对抗式与合作式
Zero-sum vs. non zero-sum	■	零和与非零和

相关的概念: Zero Sum vs. Non-zero Sum 零和与非零和博弈

- Zero sum games 零和博弈
  - Agents have *opposite utilities*.
    智能体之间是对立的方式。
  - Pure competition: win-lose, its sum is zero.
    纯竞争：输赢、其和为零。



- Non-zero sum games 非零和博弈
  - Agents have *independent*utilities.
    智能体之间是自主的方式。
  - Cooperation, indifference, competition, ...
    合作、中立、竞争、…
  - Win-win, win-loseor lose-lose, its sum is not zero.
    双赢、输赢、或双输，其和不为零。



Example: Prisoner's Dilemma 囚徒困境

- 有两个犯罪集团的成员被逮捕和监禁。每个囚徒只有二选一的机会：揭发对方并证明其犯罪，或者与对方合作保持沉默。惩罚方式如下：
- 若A和B彼此揭发对方，则每个囚徒监禁2年。
- 若A揭发B而B保持沉默，则A被释放而B监禁3年（反之亦然）。
- 若A和B都保持沉默，则他们仅被监禁1年。

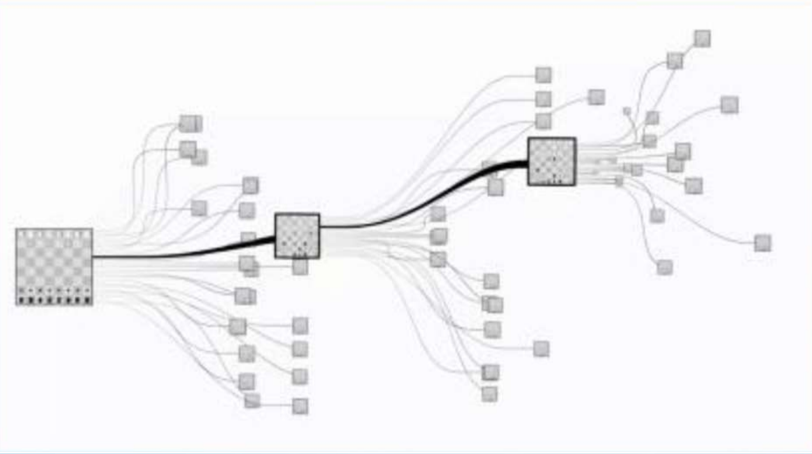
Adversarial Search often Known as Games 对抗搜索通常称为博弈

- Definitions of Game theory 博弈论的定义
  - Study of strategic decision making. Specifically, study of mathematical models of conflict and cooperation between intelligent rational decision-makers.
    研究战略决策制定。
    具体来说，研究智能理性决策者之间的冲突与合作的数学模型。
  - An alternative term is interactive decision theory.
    一个可替代的术语是交互式决策理论。
- Applications of Game theory 博弈论的应用
  - Economics, political science, psychology, logic, computer science, and biology.
    经济学、政治学、心理学、逻辑、计算机科学、以及生物学。
  - Behavioral relations and decision science, including both humans and non-humans (e.g. computers).
    行为关系与决策科学，包括人类与非人类（如计算机等）。

Games are Interesting but Too Hard to Solve 博弈有趣但难以求解

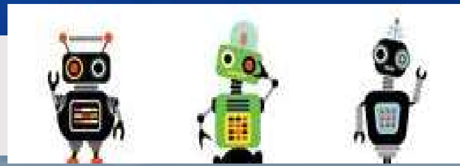
- E.g., Chess: average branching factor ≈35, each player often go to 50 moves, so search tree has about 35^{100} or 10^{154} nodes!

例如，国际象棋：平均分支数约等于35，每个对弈者常常走50多步，故该搜索树约有35^{100}或10^{154}个节点！



- Games, like the real world, therefore require the ability to make *some* decision even when calculating the *optimal* decision is infeasible.
博弈，与现实世界相似，因而当无法算计出最优决策时，需要某种决策的能力。
- Game-playing research has spawned a number of interesting ideas on how to make the best possible use of time.
博弈的研究已经产生了大量的有趣思想，即如何尽可能的利用时间。

Types of Games 博弈的类型

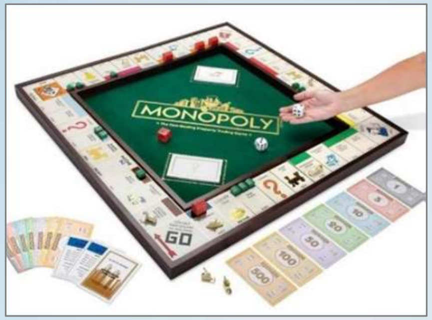


	Deterministic 确定性	Stochastic 随机性
Perfect information (fully observable) 完全信息 （可完全观测）	Chess 国际象棋 Checkers 西洋跳棋 Go 围棋 Othello 黑白棋	Backgammon 西洋双陆 Monopoly 大富翁

(a) Checkers 西洋跳棋

(b) Othello 黑白棋

(c) Backgammon 西洋双陆棋

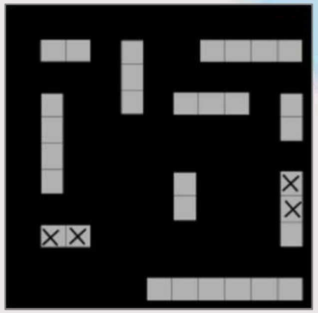


(d) Monopoly 大富翁

Types of Games 博弈的类型

	Deterministic 确定性	Stochastic 随机性
Imperfect information (partially observable) 不完全信息（部分可观测）	Stratego 西洋军棋 Battleships 海战棋	Bridge 桥牌 Poker 扑克 Scrabble 拼字游

(e) Stratego 西洋军棋



(f) Battleships 海战棋

(g) Scrabble 拼字游戏

Search vs. Adversarial Search 搜索与对抗搜索

Search 搜索	**Adversarial Search** 对抗搜索
Single agent 单智能体	Multiple agents 多智能体
Solution is (heuristic) method for finding goal. 解是寻找目标的（启发式）方法	Solution is strategy (strategy specifies move for every possible opponent reply). 解是策略（指定对每个可能对手回应的行动策略）
Heuristics can find optimal solution. 启发式法可以找到最优解	Time limits force an approximate solution. 时间受限被迫执行一个近似解
Evaluation function: estimate of cost from start to goal through given node. 评价函数：给定节点从起始到目标的代价估计	Evaluation function: evaluate “goodness” of game position. 评价函数：评估博弈局势的“好坏”

特定环境

deterministic, perfect information, turn taking, two players, zero sum

确定、完全可观察、轮流、双人、零和（Ps. 通常是有时间约束的）

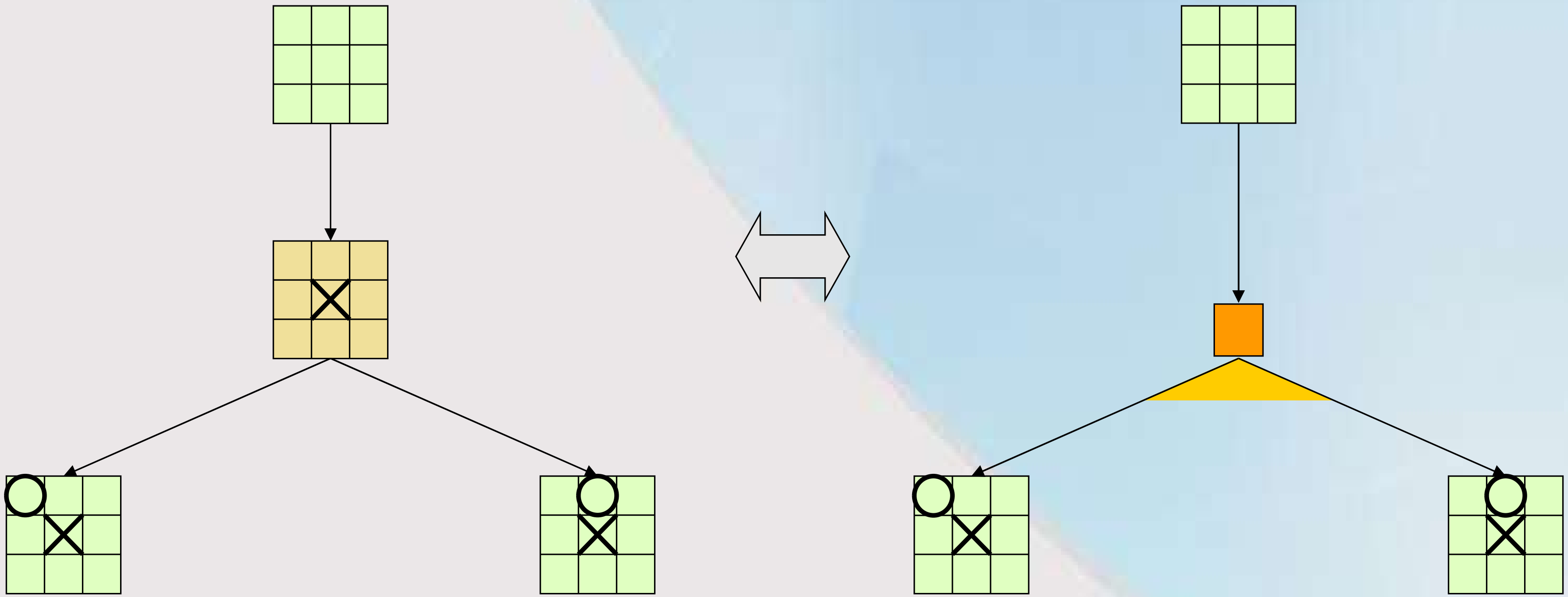
- Calling the two players:
  将两个玩家称为：**MAX, MIN.**
- **MAX** moves first, and then they take turns moving, until the game is over.
  MAX先走棋，然后轮流走棋，直到博弈结束。
- At game end 博弈结束时
  - winner: award points
    胜者：奖励点数
  - loser: give penalties.
    败者：给予处罚

Formally Defined as a Search Problem 形式化定义为搜索问题

S_0	▪ Initial state, specifies how the game is set up at the start. 初始状态，指定博弈开始时的设定。
PLAYER(*s*)	▪ Defines which playerhas the move in a state. 定义哪个玩家在某状态下动作。
ACTIONS(*s*)	▪ Returns the set of legal moves in a state. 返回某个状态下的合法动作。
RESULT(*s*, *a*)	▪ Transition model, defines the result of a move. 转换模型，定义一步动作的结果。
TERMINAL-TEST(*s*)	▪ Terminal test, *true*when the game is over and *false* otherwise. 终止检测，博弈结束时为*true*，否则为*false*。
UTILITY(*s*, *p*)	▪ Utility function, defines the value in states for a player *p*. 效用函数，定义在状态*s*、玩家为*p*的值。

相关说明

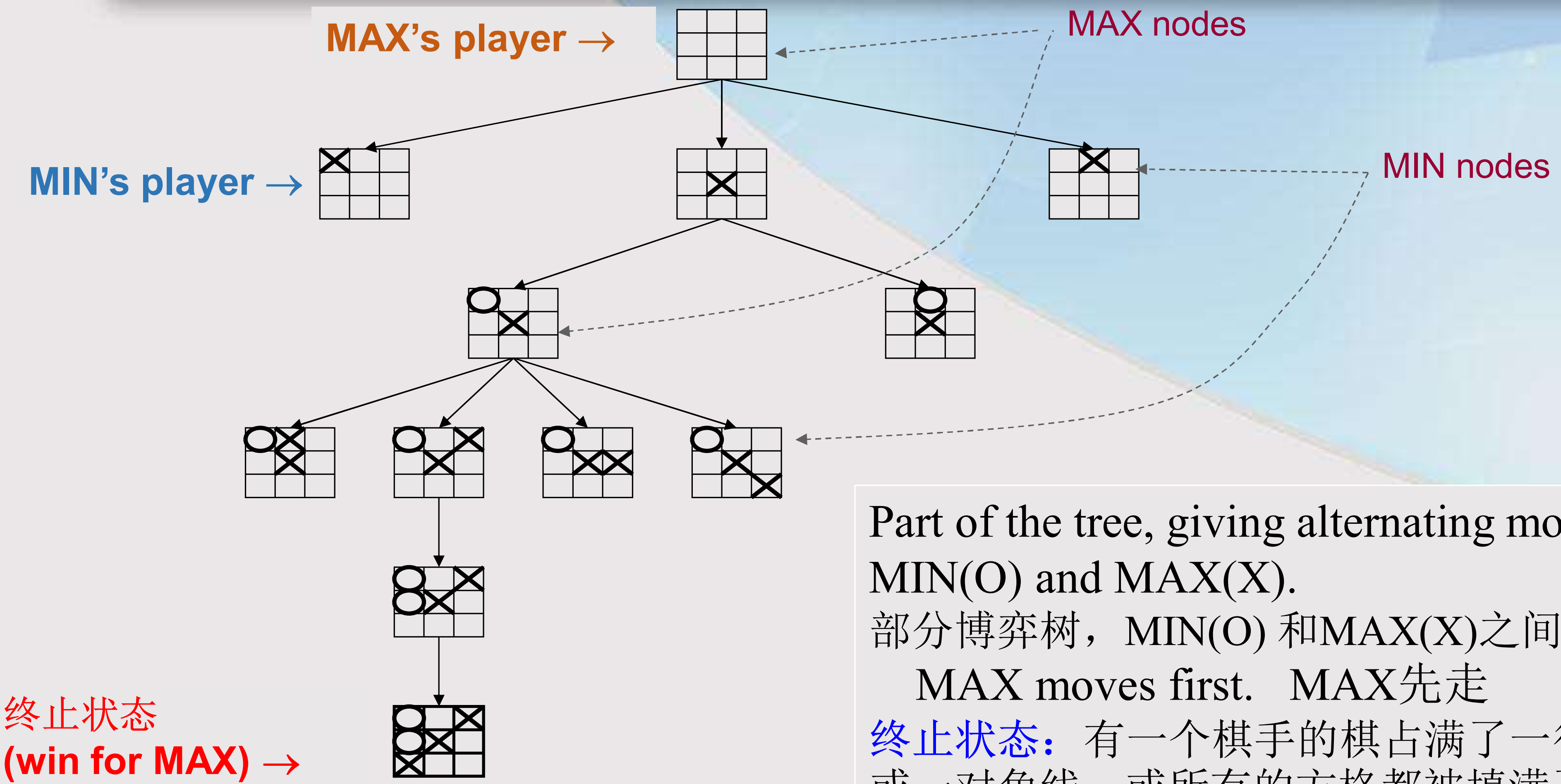
- MAX的不确定性是由另一个智能体MIN（对手）的行为引起的。



相关说明

- MAX的不确定性是由另一个智能体MIN（对手）的行为引起的。
- 双方均要对方失败。
- 不存在保证MAX胜利的计划，原因是MAX的后继状态由MIN的行为决定（对MIN也是同样）
- 在每次换手时，有一个特定的时间限制
- 状态空间是巨大的：在规定时间内，仅一个空间的小片段可以探究。

Example: Game Tree of Tic-tac-toe 井字棋的博弈树

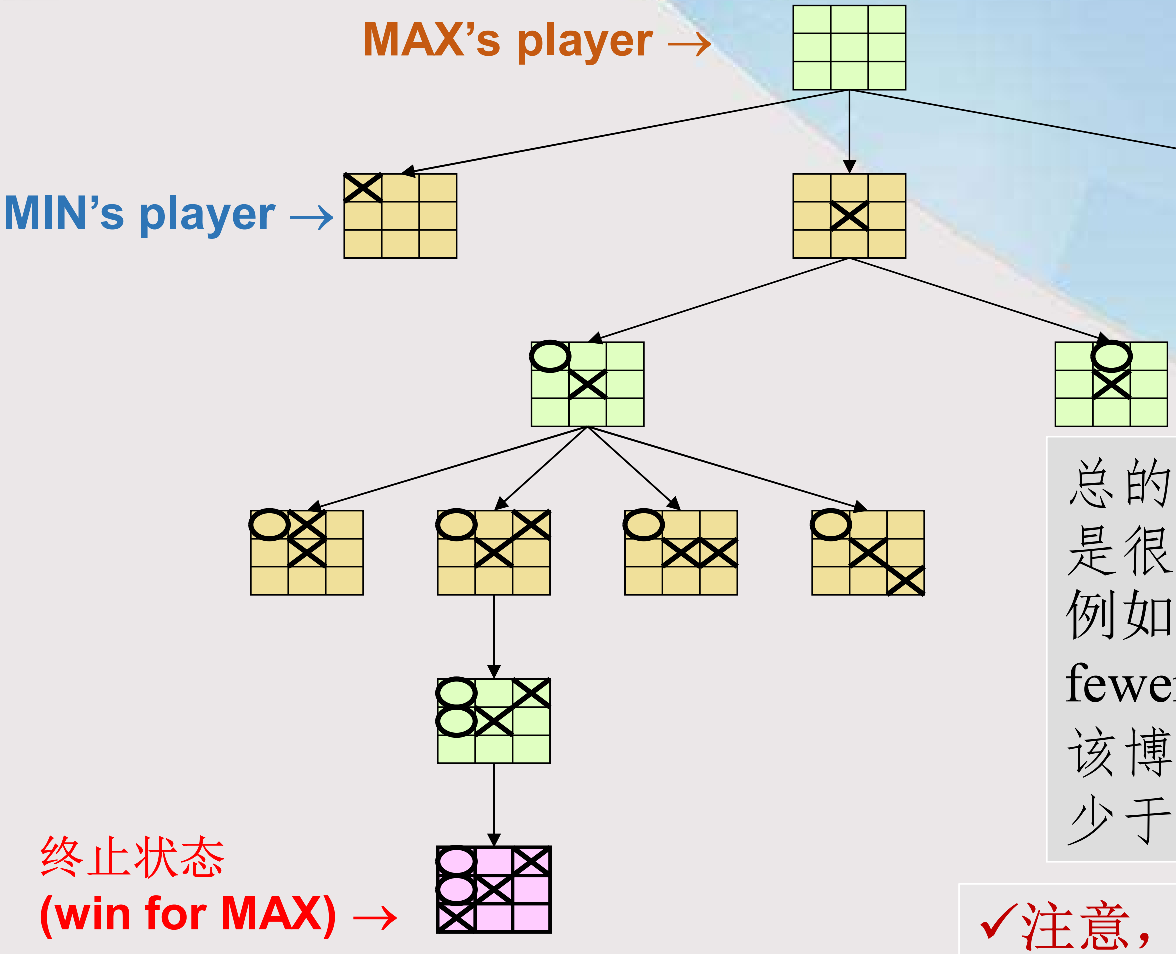


Part of the tree, giving alternating moves by MIN(O) and MAX(X).
部分博弈树，MIN(O) 和MAX(X)之间交替走子

MAX moves first. MAX先走

终止状态：有一个棋手的棋占满了一行，一列，或一对角线，或所有的方格都被填满了。

Example: Game Tree of Tic-tac-toe 井字棋的博弈树



总的说来，分支因子和终止状态的深度是很大的。
例如：The game tree is relatively small, fewer than 9! = 362,880 nodes.
该博弈树相对较小，
少于9! = 362,880个节点

✓注意，这里利用对称性来减少分支因子。

目 录

Zhaoman, AI,
Department of Computer Science, CS, CUG

Optimal Solution 最优解

- In normal search 普通搜索
  - The optimal solution would be a sequence of actions leading to a goal state(terminal state) that is a win.
    最优解将是导致获胜的目标状态（终端状态）的一系列动作。
- In adversarial search 对抗搜索
  - Both of MAX and MIN could have an optimal strategy.
    MAX和MIN都会有一个最优策略。
    - ✓In initial state, MAXmust find a strategy to specify MAX's move,
      在初始状态，MAX必须找到一个策略来确定MAX的动作，
    - ✓then MAX's moves in the states resulting from every possible response by MIN, and so on.
      然后MAX针对MIN的每个合理的对应采取相应的动作，以此类推。

Minimax Theorem 最小最大定理

For every two-player, zero-sum game with finitely many strategies, there exists a value V and a mixed strategy for each player, such that

对于两个玩家、具有有限多个策略的零和博弈，每个玩家存在一个值V和一个混合策略，使得：

(a) Given player 2’s strategy, the best payoff possible for player 1 is V,

给定玩家2的策略，则玩家1可能的最好收益是V，

(b) Given player 1's strategy, the best payoff possible for player 2 is −V.

给定玩家1的策略，则玩家2可能的最好收益是-V。

- For a zero sum game, the name **minimax** arises because each player *minimizes the maximum payoff* possible for the other, he also *minimizes his own maximum loss*.
  对于零和博弈来说，其名称**minimax**的由来是因为每个玩家会使对手可能的最大收益变得最小，还会使自己的最大损失变得最小。

Optimal Solution inAdversarial Search 对抗搜索的最优解

- Given a game tree, the optimal strategy can be determined from the minimax valueof each node, write as MINIMAX(*n*).

  给定一棵博弈树，则最优策略可以由每个节点的minimax值来确定，

  记作 **MINIMAX(n)**。

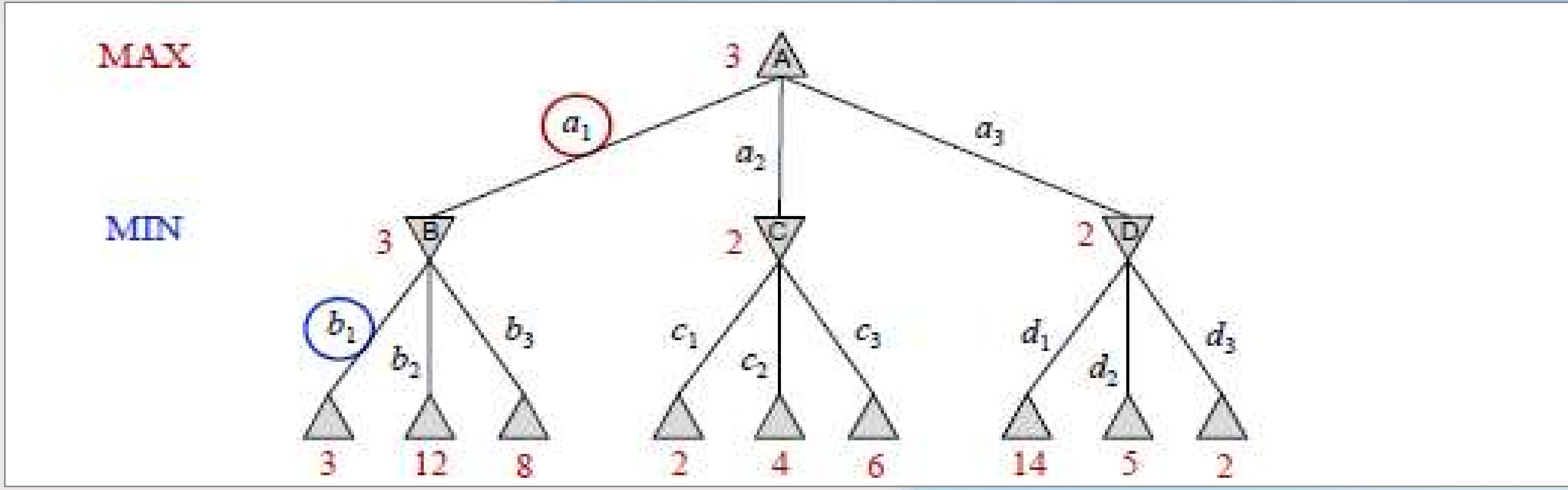
- Assume that both players play optimally from there to the end of the game.

  假设两个玩家博弈自始至终都发挥得很好。

function MINIMAX(s) **returns** an action
if TERMINAL-TEST(s) **then return** **UTILITY(s)**
If PLAYER(s) = MAX **then return** $\max_{a \in \text{ACTIONS}(s)}$ MINIMAX(RESULT(s, a))
if PLAYER(s) = MIN **then return** $\min_{a \in \text{ACTIONS}(s)}$ MINIMAX(RESULT(s, a))

The minimax value of a terminal state is just its utility.
MAX prefers to move to a state of maximum value, MIN prefers a state of minimum value.
终端状态的**minimax**值只是其效用。**MAX**倾向于移动到一个最大值状态，**MIN**则倾向于一个最小值状态。

Minimax Decision--A Two-player Game Tree 一个双人玩家的博弈树



MAX's best move at root is a_1(with the highest minimax value)

根节点处MAX的最佳移动是a_1（具有最高的minimax值）

MIN's best reply at B is b_1(with the lowest minimax value)

B节点处MIN的最佳应对是b_1（具有最低的minimax值）

博弈中的优化决策：评估函数

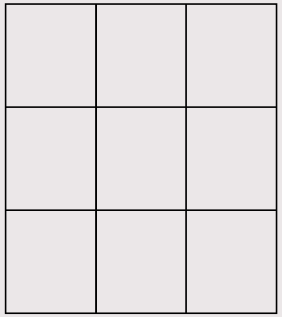
- 用函数e(s)代表状态s下的值；
- e(s) 是MAX状态的启发值，是对MAX有利状态的估计；
- e(s)>0 代表对MAX有利，值越大越好；
- e(s)<0 代表对MAX不利；
- e(s)=0 代表对MAX、MIN势均力敌。

例子： Tic-tac-Toe 井字棋

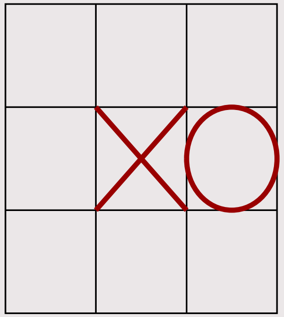
e(s) = 在所有空位放上MAX后成3子一线的个数

减去-

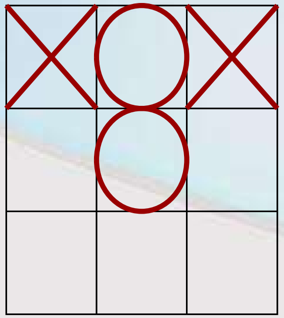
在所有空位放上MIN后成3子一线的个数



8–8 = 0



6–4 = 2



3–3 = 0

评估函数的组成

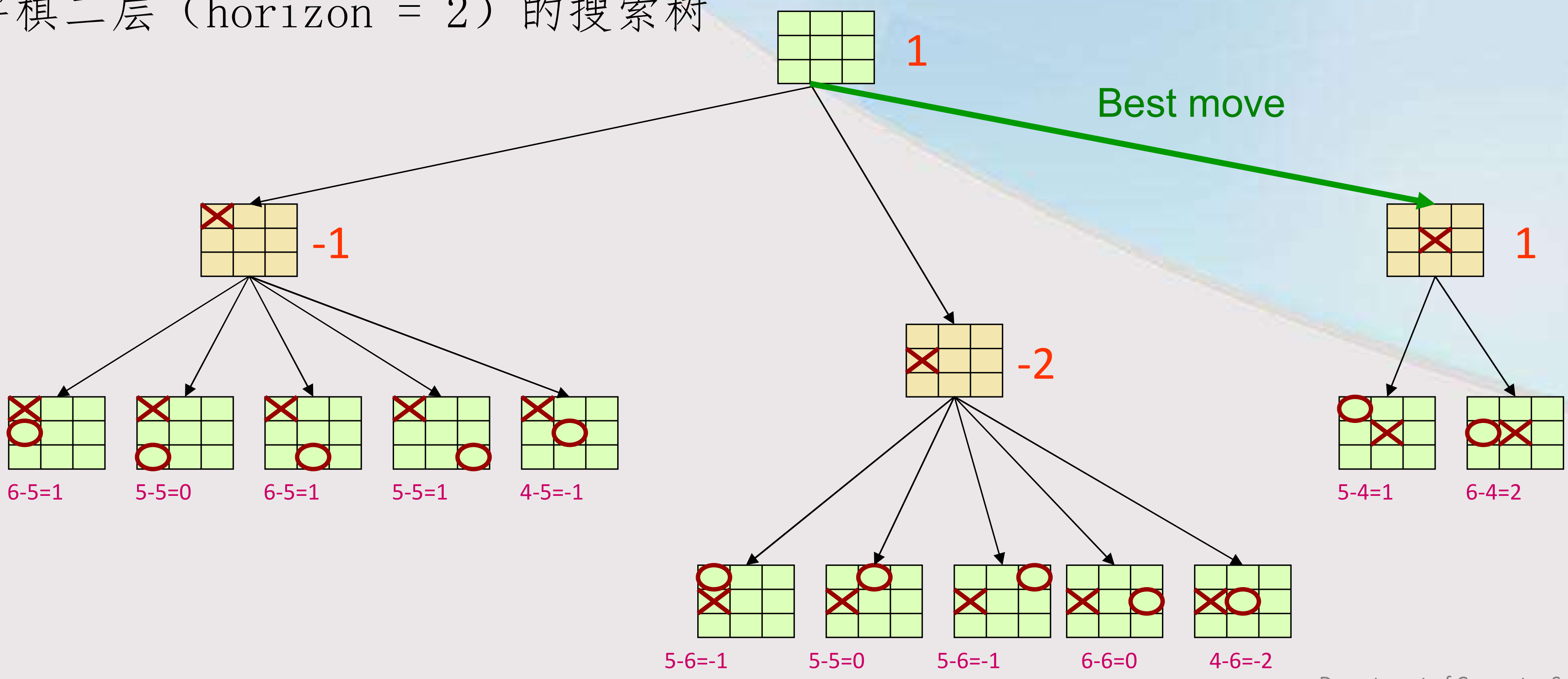
- 各“特征值”的总和：

$$e(s)=\sum_{i=1}^{n} w_i f_i(s)$$

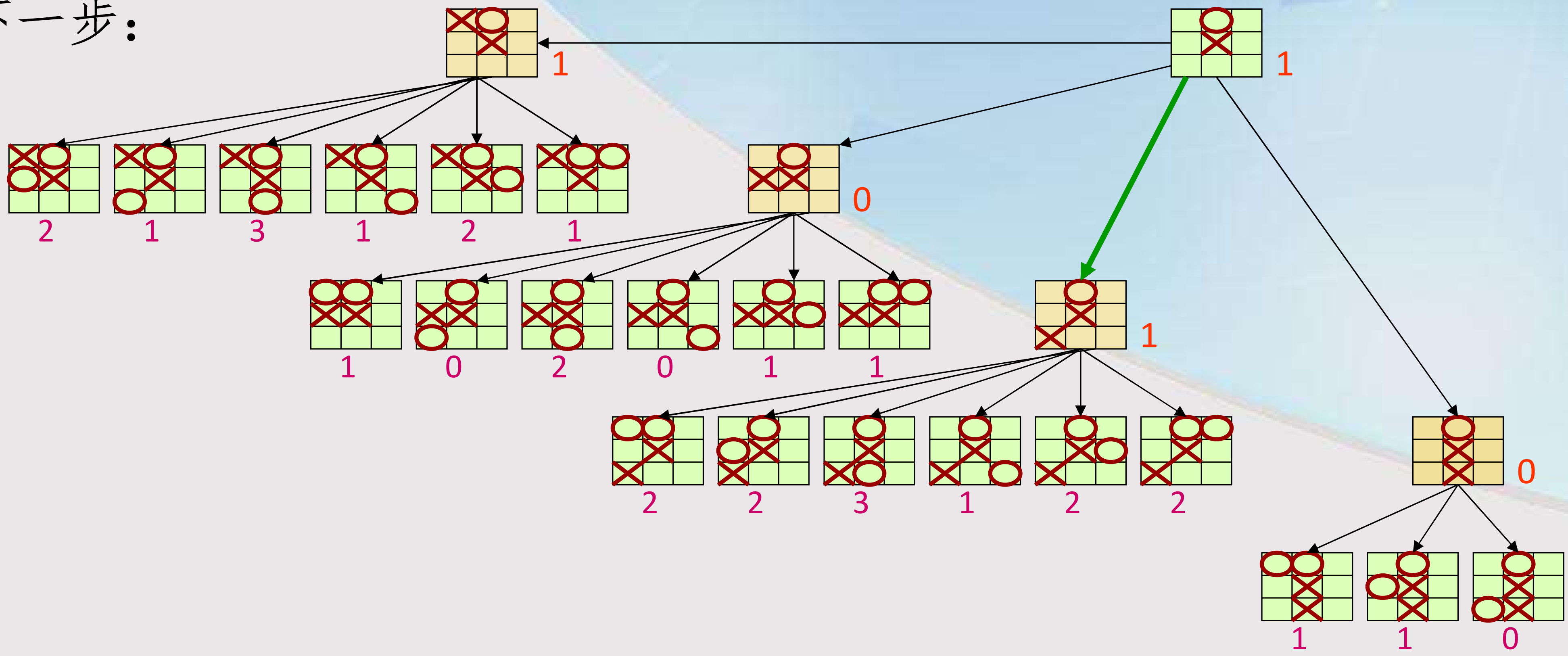
- 特征值包括：
  - 每种势态的数值
  - 可能移动的数值
  - 势态的控制数值

评估值：回退计算

井字棋二层（horizon = 2）的搜索树



下一步：



为什么采用回退值?

1.对每一个非叶子结点N的评估值是由它的子结点倒推求得，是MAX能搜索到的深度h回推得到最安全状态下最佳走步（假设MIN是足够好的棋手）。

2.如果从一开始e就是可信赖的，那么实际状态的值是要好于评估函数的值的。

极大极小算法思想 Minimax Algorithm

- 轮到**MAX**走时，从当前状态扩展博弈树到所设置深度h（根据允许时间要求所能的深度）；
- 计算每个叶子节点的评估值；
- 根据不同的叶子节点采用不同的方法倒推计算各节点值：
  - 对于MAX节点选取其后继节点中最大的值为其评估值；
  - 对于MIN节点选取其后继节点中最小的值为其评估值。
- 选择移动到具有最大倒推值的**MIN**节点。

Game Playing (for MAX)

重复直到达到终止条件（胜、负、平）：

1. 用**MINIMAX**算法选择移动；
2. 执行移动；
3. 观察**MIN**的移动。

注意：每次循环产生的深度为h的大博弈树仅为一次移动而进行；
所有的下次循环将被再次重复。
（其实一个深度h减2的子树可以被再次利用）。

Minimax Algorithm 最小最大算法

function MINIMAX-DECISION(*state*) returns *an action*
  inputs: *state*, current state in game
  $v \leftarrow$ MAX-VALUE(*state*)
  return the *action* in SUCCESSORS(*state*) with value *v*

---

function MAX-VALUE(*state*) returns *a utility value*
  if TERMINAL-TEST(*state*) then return UTILITY(*state*)
  $v \leftarrow -\infty$
  for *a,s* in SUCCESSORS(*state*) do
    $v \leftarrow$ MAX(*v*,MIN-VALUE(*s*))
  return *v*

---

function MIN-VALUE(*state*) returns *a utility value*
  if TERMINAL-TEST(*state*) then return UTILITY(*state*)
  $v \leftarrow \infty$
  for *a,s* in SUCCESSORS(*state*) do
    $v \leftarrow$ MIN(*v*,MAX-VALUE(*s*))
  return *v*

Properties of Minimax Decision 最小最大决策的性质

- The minimax algorithm performs a depth-firstexploration of the game tree.

  最小最大算法表现为博弈树的深度优先探索。

  - Time complexity 时间复杂性 $O(b^m)$
  - Space complexity 空间复杂性

- ***O*(*bm*)** --The algorithm generates all actions at once 算法同时生成所有动作

- ***O*(*m*)** --The algorithm generates actions one at a time 算法一次生成一个动作

➢ Where *b*--The branching factor (legal moves at each point)

分支因子（每个点的合法走子）

m--The maximum depth of any node

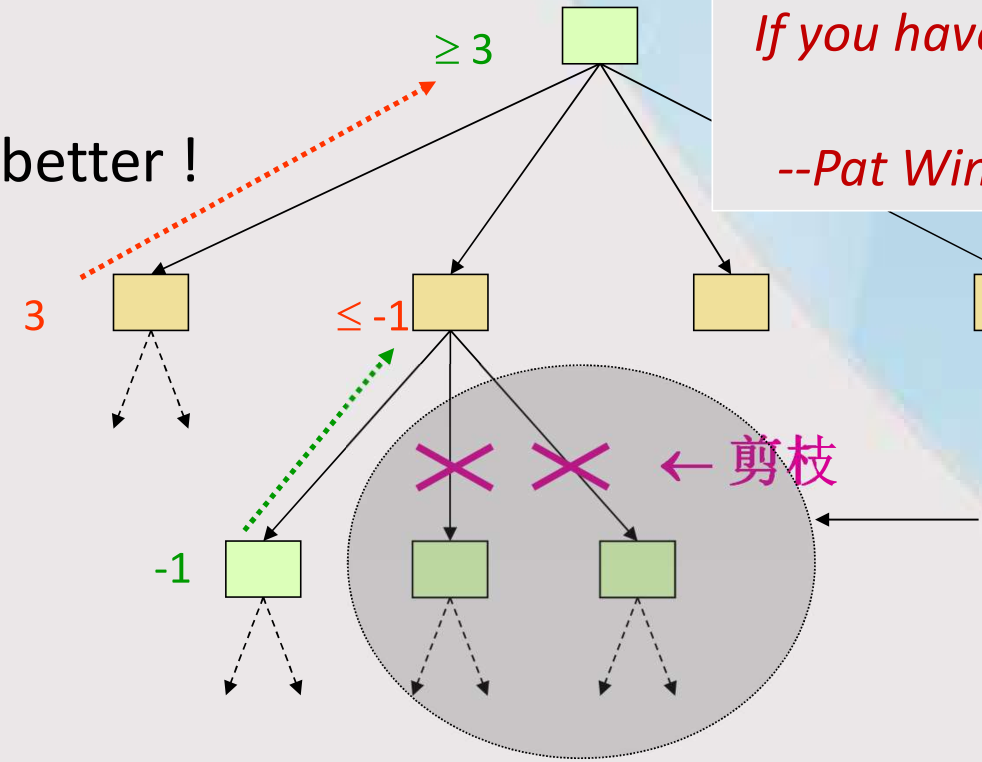
任一节点的最大深度

我们可以做的更好么?

Number of game states is exponential in depth of the tree.
博弈状态的量随着树的深度呈现指数式增长。

If you have an idea that is surely bad, don't take the time to see how truly awful it is.
--Pat Winston (Director, MIT AI Lab,1972-1997)

Yes ! Much better !



搜索树的这一部分对倒推求值没有任何影响。
我们把这种对不需要生成的节点不去生成的方法称为剪枝

目 录

Overview 概述

- The trick to solve the problem:
  解决该问题的技巧
- Compute correct minimax decision without looking at every node in game tree.
  计算正确的minimax决策而不考虑博弈树的每个节点。
- That is, use“pruning” to eliminate large parts of the tree.
  就是说，采用“剪枝”方法来消除该树的大部分。
- What is alpha–beta pruning
  什么是alpha–beta剪枝
- It is a search algorithm that seeks to decrease the number of nodes that are evaluated by the minimax algorithm.
  是一种搜索算法，旨在削减由minimax算法评价的节点数量。

Why Called Alpha-Beta 为何称其为Alpha-Beta

- Alpha–beta pruning gets its name from the following two parameters:

  Alpha-Beta剪枝从如下两个参数得到其名称：

  ✓ α:highest-value we have found so far at any point along the path for MAX.

  α：沿着MAX路径上的任意选择点，迄今为止我们已经发现的最高值。

  ✓ β:lowest-value we have found so far at any point along the path for MIN.

  β：沿着MIN路径上的任意选择点，迄今为止我们已经发现的最低值。

- Alpha–beta search respectively:

  Alpha-Beta搜索依次完成如下动作：

  ✓ updates the values of α and β as it goes along, and

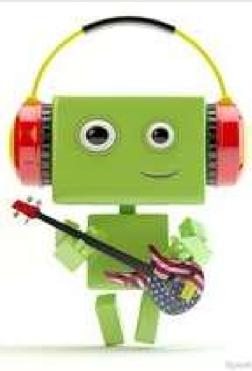
  边搜索边更新α和β的值，并且

  ✓ prunes the remaining branches at a node as soon as the value of the current node is known to be worse than the current α or β value for MAX or MIN.

  一旦得知当前节点的值比当前MAX或MIN的α或β值更差，则在该节点剪去其余的分枝。

China University of Geosciences

Alpha-Beta 搜索算法

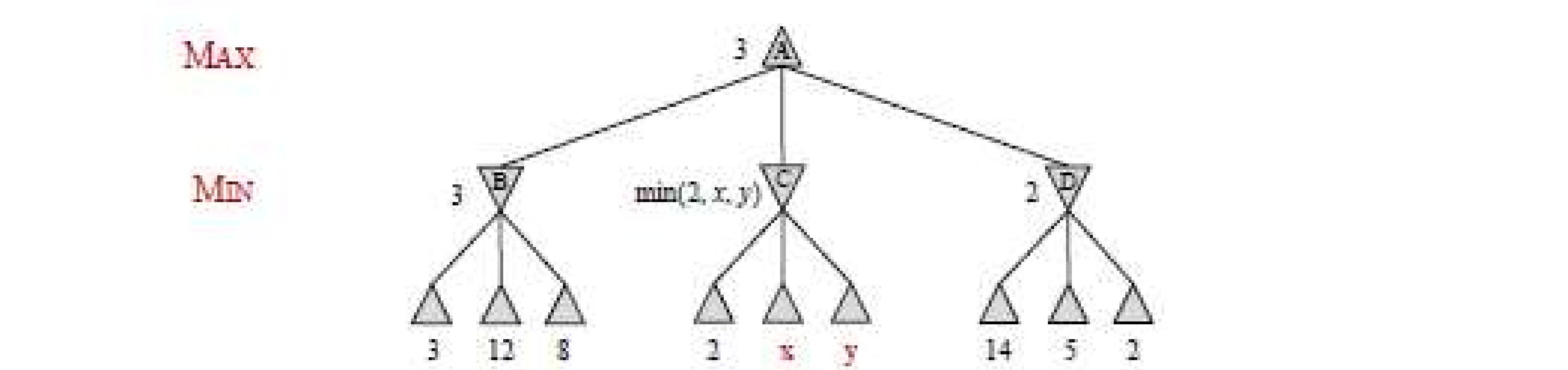


function ALPHA-BETA-SEARCH(*state*) **returns** an action
$v \leftarrow$ MAX-VALUE(*state*, $-\infty$, $+\infty$)
Return the *action* in ACTIONS(*state*) with value v

function MAX-VALUE(*state*, α, β) **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(*state*, *a*), α, β))
if $v \geq \beta$ **then return** v **else** $\alpha \leftarrow$ MAX(α, v)   // β剪枝
return v

function MIN-VALUE(*state*, α, β) **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(*state*, *a*), α, β))
if $v \leq \alpha$ **then return** v **else** $\beta \leftarrow$ MIN(β, v)   // α剪枝
Return v

Example: Game Tree Using Minimax 采用Minimax的博弈树

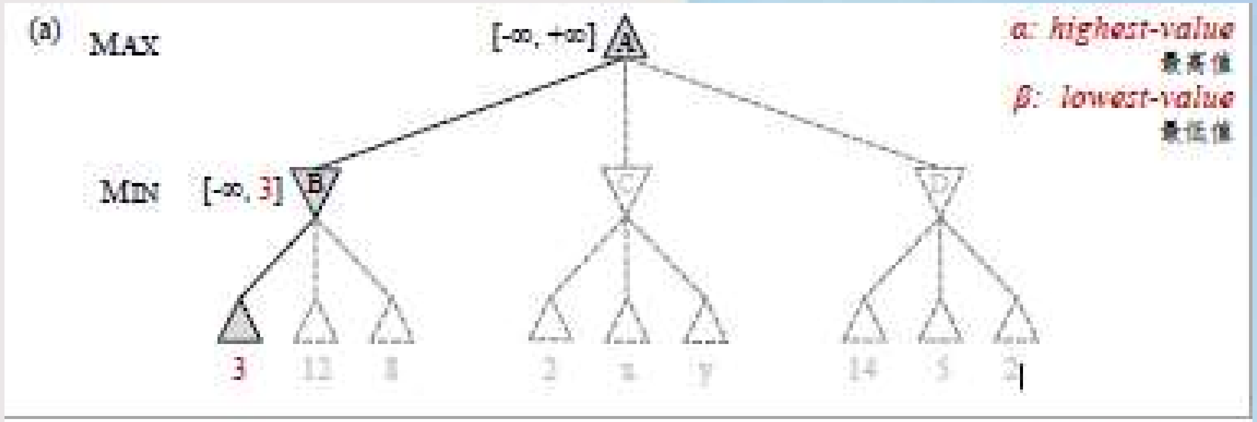


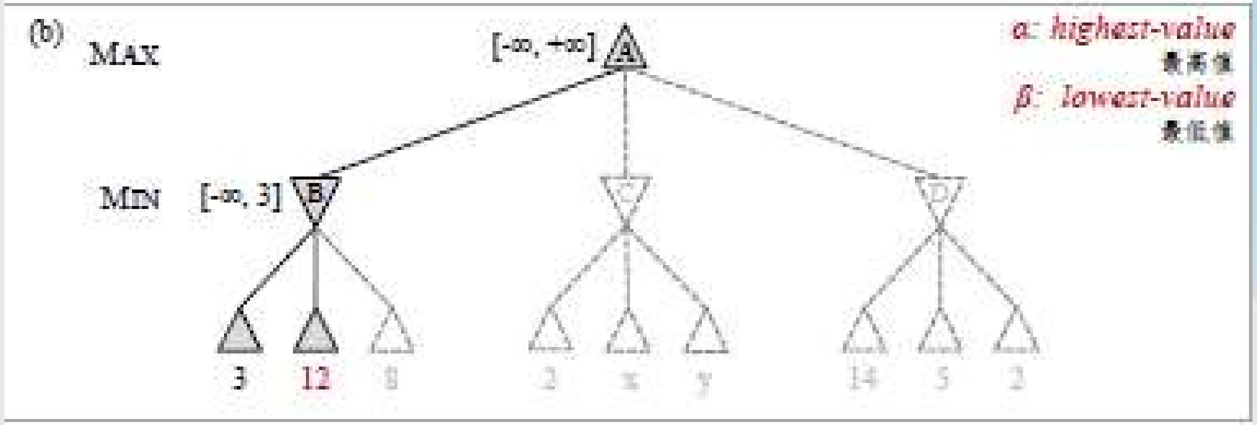
- The value of root node is given by: 根节点的值由如下方法得出：

$$
\begin{aligned}
\text{MINIMAX}(\mathit{root}) &= \max(\min(3, 12, 8), \min(2, x, y), \min(14, 5, 2)) \\
&= \max(3, \min(2, x, y), 2) \\
&= \max(3, z, 2) \text{ where } z = \min(2, x, y) \leq 2 \\
&= 3.
\end{aligned}
$$

no pruning!

Example: Game Tree Using Alpha-Beta Pruning
采用Alpha-Beta剪枝的博弈树





Initial value: 初始值：

$A[\alpha=-\infty, \beta=+\infty]$

(a) The 1st leaf below B has the value 3. Hence, B, as a MIN node,

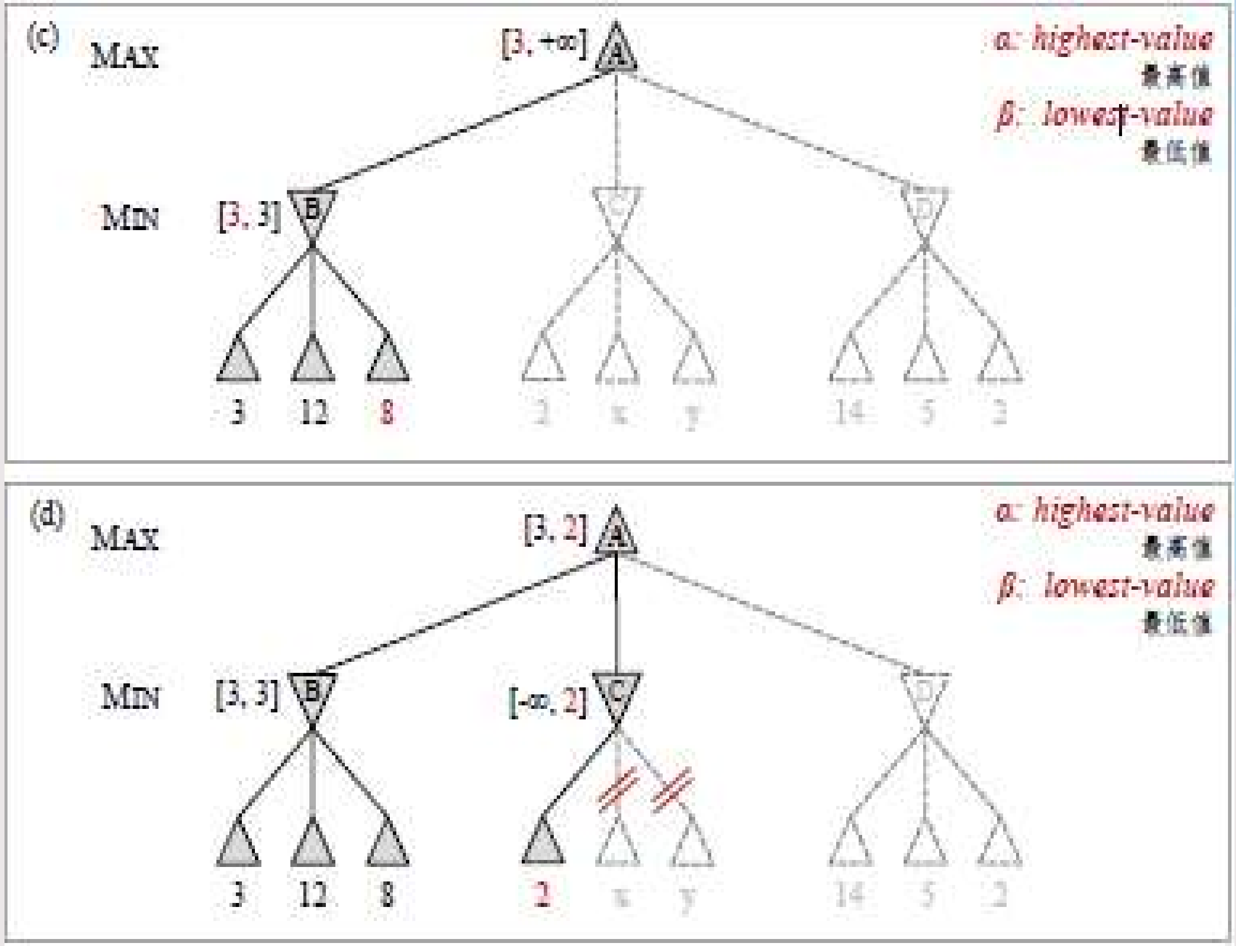
$B[\beta=3]$.

(b) The 2nd leaf below B has the value 12;MIN would avoid this move,

still,

$B[\beta=3]$.

Example: Game Tree Using Alpha-Beta Pruning 采用Alpha-Beta剪枝的博弈树



(c) The 3rd leaf below B has a value of 8; so exactly MIN node $B[\beta=3]$.

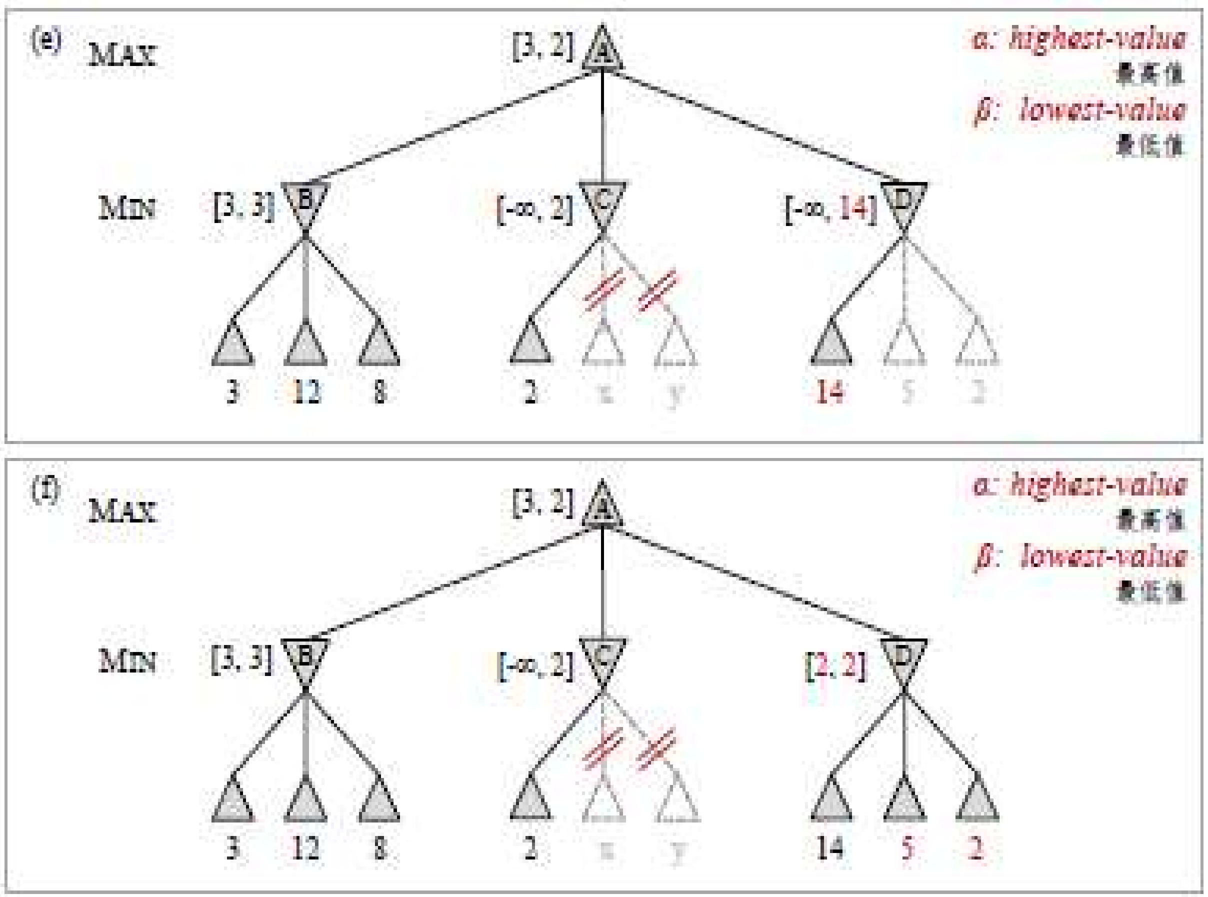
Now, we can infer B[α=3], because MAX has A[α≥3].

B下面第三个叶节点的值为8；故MIN节点正是$B[\beta=3]$。现在，因为MAX为$A[\alpha\geq3]$，我们能够推出$B[\alpha=3]$。

(d) The 1st leaf below C has the value 2, hence, as a MIN node $C[\beta=2]$, and $B[\beta=3]>C[\beta=2]$, so MAX would never choose C. Therefore just prune all successor of C(α–β pruning).

C下面第一个叶节点的值为2，因此，由于MIN节点$C[\beta=2]$，且$B[\beta=3]>C[\beta=2]$，故MAX将不会选择C，所以只需剪掉C的所有后继节点(α–βpruning)。

Example: Game Tree Using Alpha-Beta Pruning 采用Alpha-Beta剪枝的博弈树



(e) The 1st leaf below D is 14, $D[\beta\leq14]$, so we need to keep exploring D's successor states. We now have bounds on all of root's successors, so $A[\beta\leq2]$.

D下面第一个叶节点为14，$D[\beta\leq14]$，故我们需要不断搜索D节点的后继状态。到此我们已经遍布了根节点的所有后继节点，故$A[\beta\leq2]$。

(f) The 2nd successor of D is worth 5, so keep exploring. The 3rd successor is worth 2, so $D[\beta=2]$. MAX's decision at the root keeps $A[\beta=2]$.

D的第2个后继节点的值等于5，故不断搜索，第3个后继节点等于2，故$D[\beta=2]$。

根节点MAX的抉择保持$A[\beta=2]$。

General Principle of Alpha-Beta Pruning α - β 剪枝的一般原则

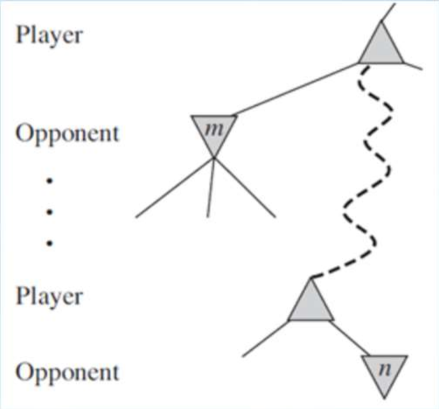
- Alpha–beta pruning can be applied to trees of any depth, and often possible to prune entire subtrees rather than just leaves.

α - β剪枝可被用于任意深度的树，并且常常可以剪去整个子树而不仅仅是叶节点。

- The general principle: 一般原则：
  - Consider a node n somewhere in the tree, such that Player has a choice of moving to that node.

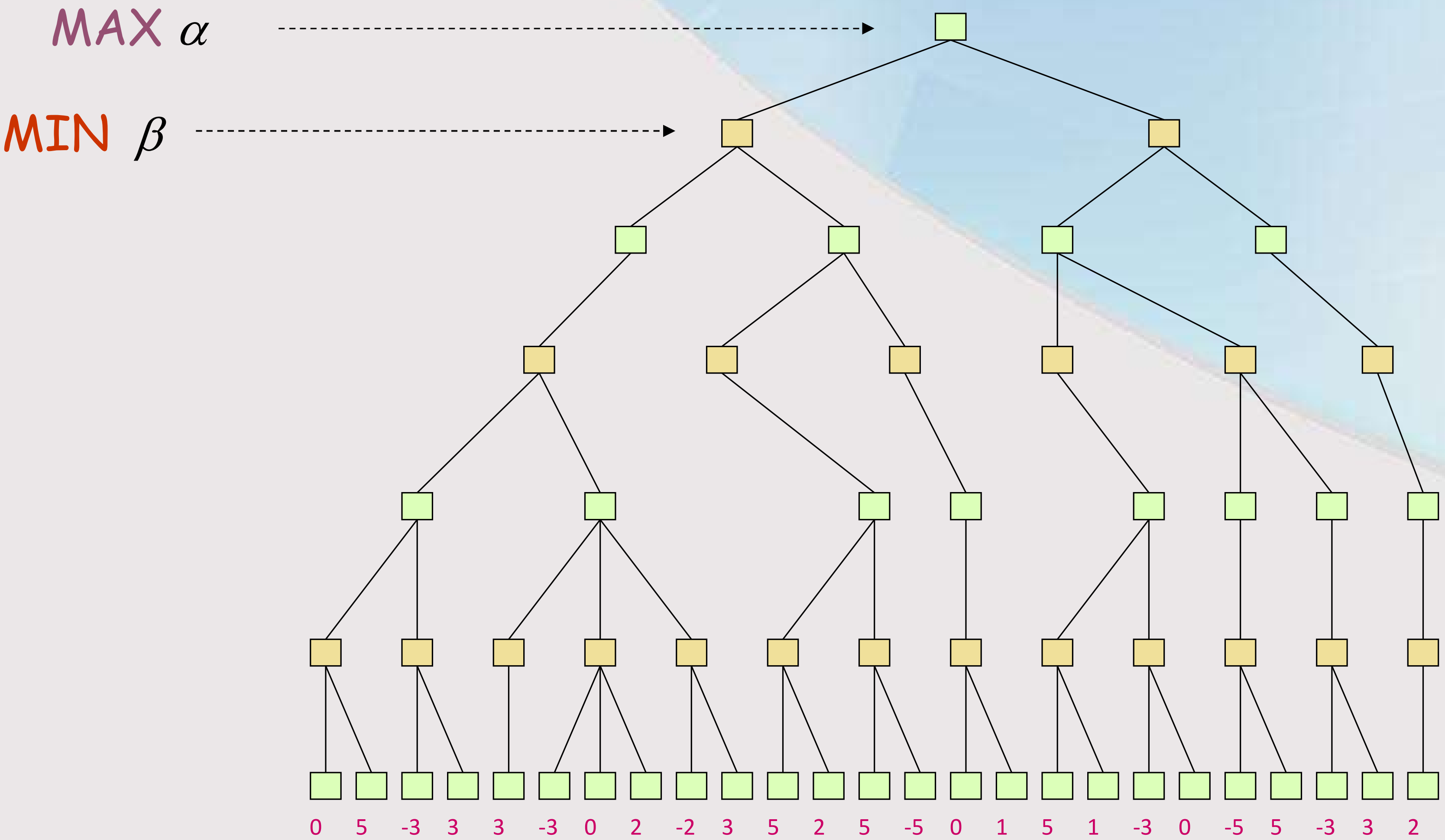
    设某个节点n位于树的某处，于是玩家选择移向那个节点。
  - If Player has a better choice $m$at parent node of n, or at any choice point further up, then n *will never be reached* in actual play.

    若玩家在位于n的父节点或更上层处有更好的选择m，则在实战中 完全没必要抵达n。



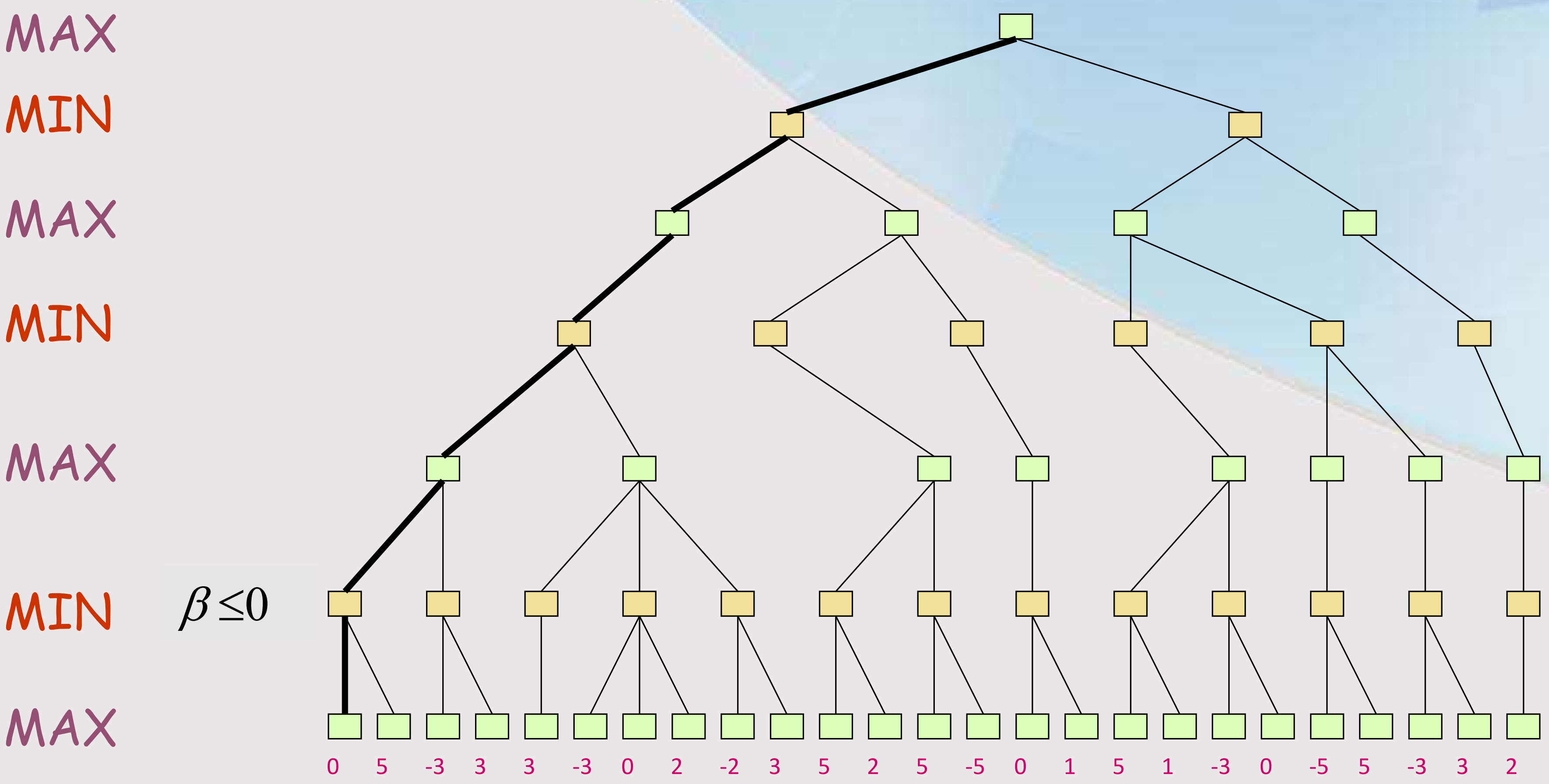
Zhaoman, AI,
Department of Computer Science, CS, CUG

课堂练习：以α-β 算法思想搜索，并标明何处发生何种剪枝

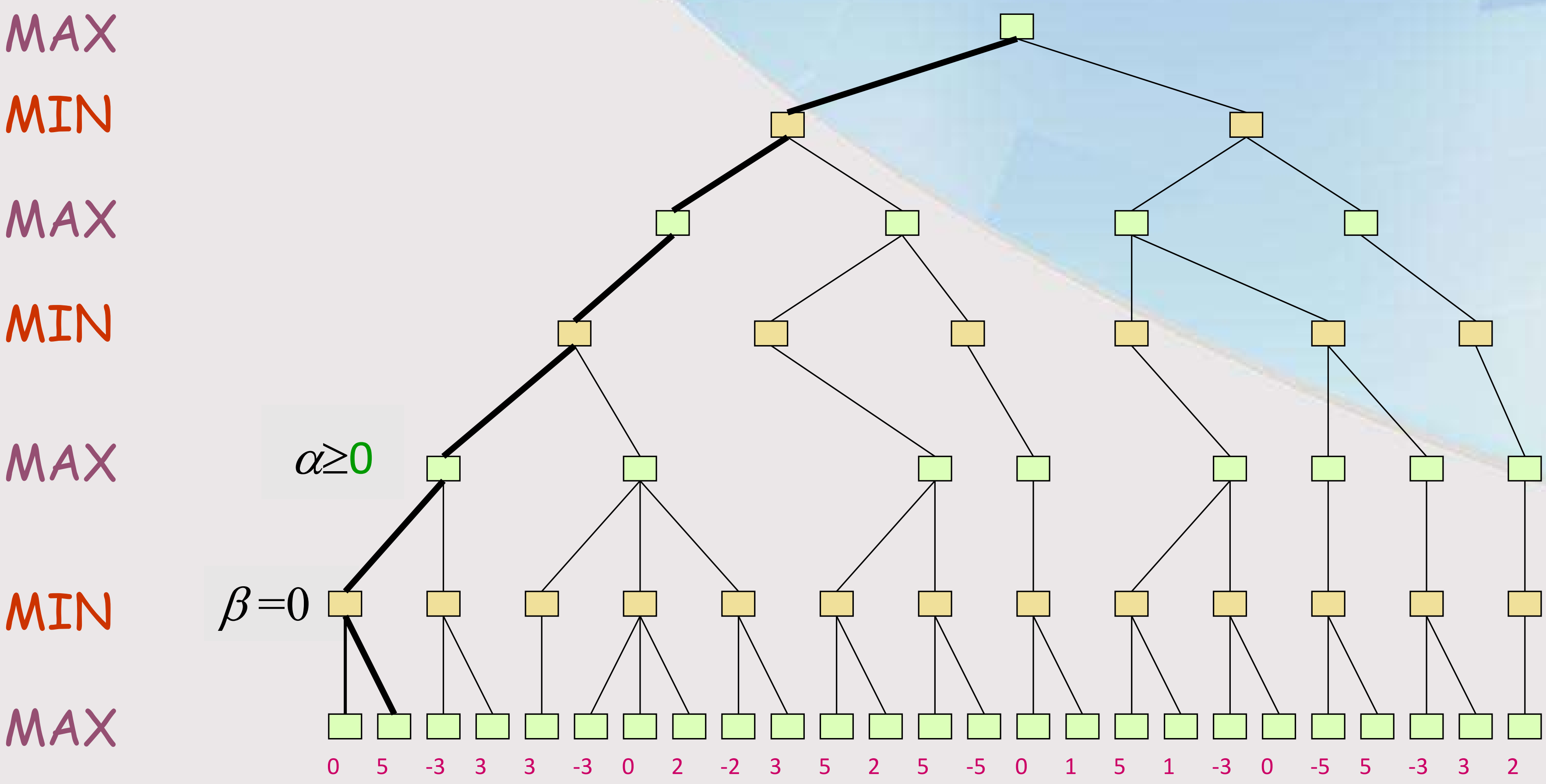


Zhaoman, AI,
Department of Computer Science, CS, CUG

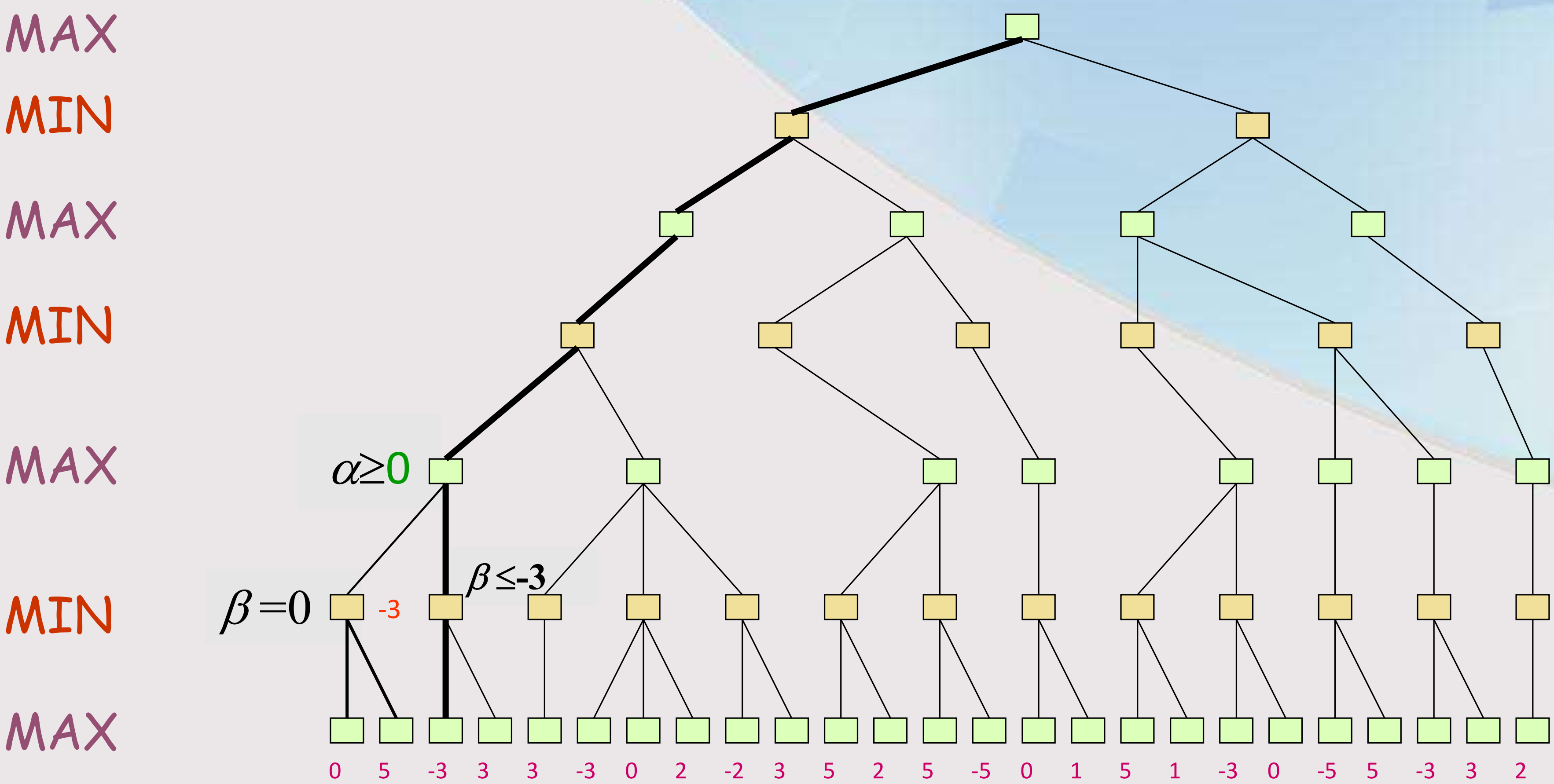
Example


MAX
MIN
MAX
MIN
MAX
MIN
MAX
$\beta \leq 0$
0 5 -3 3 3 -3 0 2 -2 3 5 2 5 -5 0 1 5 1 -3 0 -5 5 -3 3 2

Example

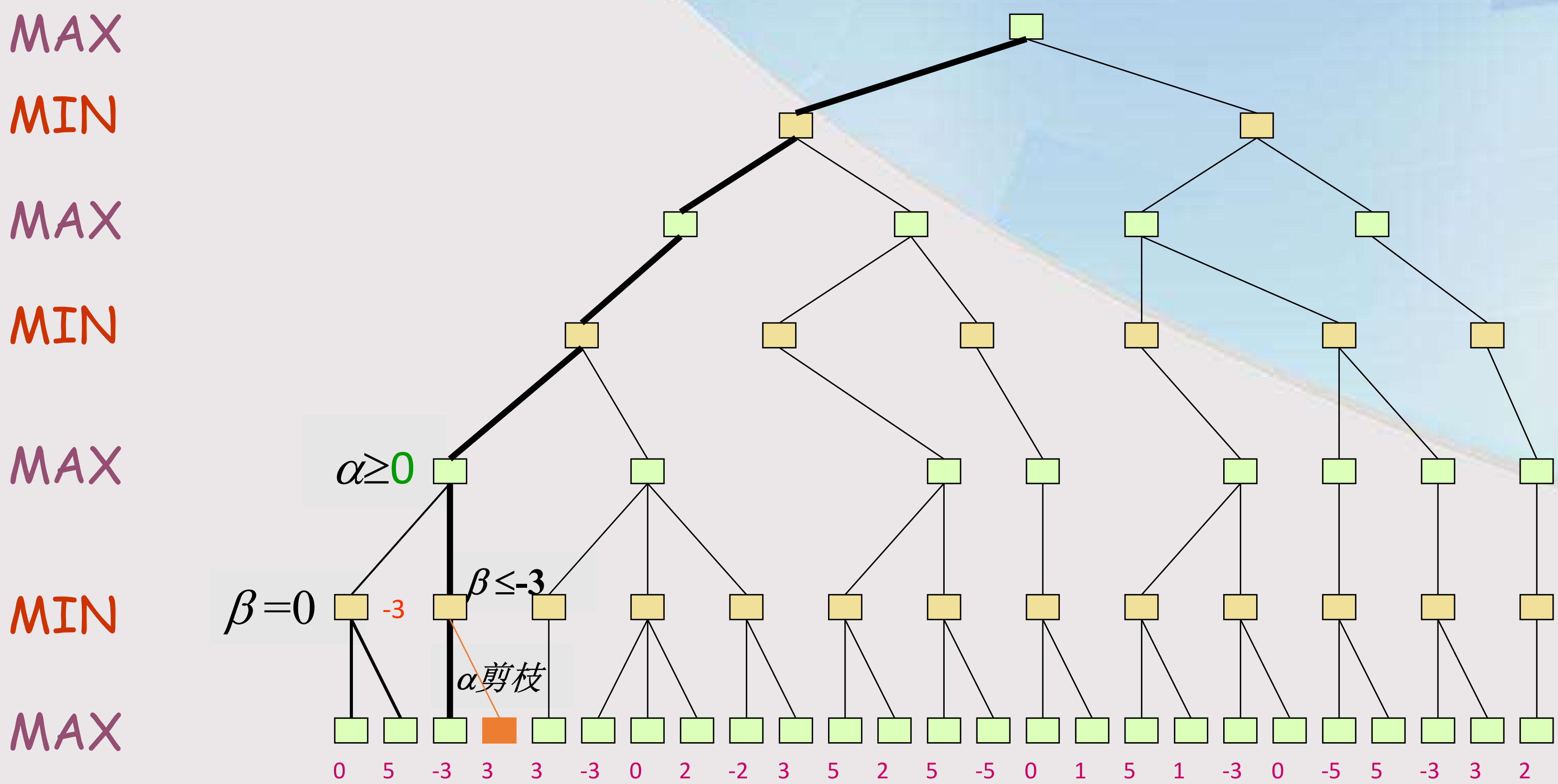

MAX
MIN
MAX
MIN
MAX
MIN
MAX
$\alpha \geq 0$
$\beta = 0$
0 5 -3 3 3 -3 0 2 -2 3 5 2 5 -5 0 1 5 1 -3 0 -5 5 -3 3 2

Example

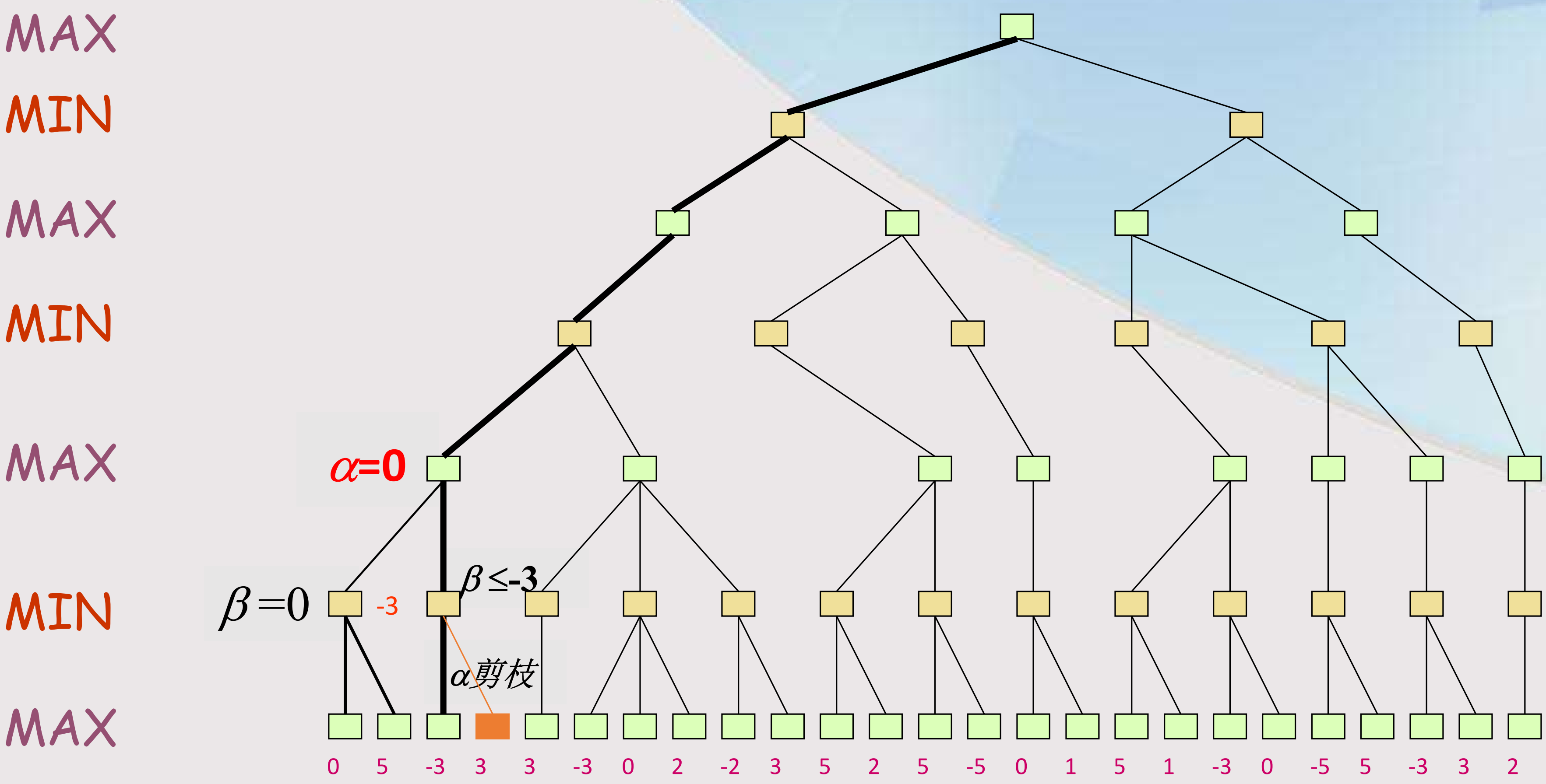


MAX
MIN
MAX
MIN
MAX
MIN
MAX
α≥0
β=0
-3
β≤-3
0 5 -3 3 3 -3 0 2 -2 3 5 2 5 -5 0 1 5 1 -3 0 -5 5 -3 3 2

Example


MAX
MIN
MAX
MIN
MAX
MIN
MAX
α≥0
β=0
-3
β≤-3
α剪枝
0 5 -3 3 3 -3 0 2 -2 3 5 2 5 -5 0 1 5 1 -3 0 -5 5 -3 3 2

Example


MAX
MIN
MAX
MIN
MAX
MIN
MAX
α=0
β=0
-3
β≤-3
α剪枝
0 5 -3 3 3 -3 0 2 -2 3 5 2 5 -5 0 1 5 1 -3 0 -5 5 -3 3 2

作业：以α-β 算法思想搜索，并标明何处发生何种剪枝
搜索从右至左进行。