人工智能：搜索技术 III

饶洋辉
计算机学院,
中山大学
raoyangh@mail.sysu.edu.cn
http://cse.sysu.edu.cn/node/2471

课件来源：中山大学刘咏梅教授；北京大学王文敏教授；浙江大学吴飞教授；多伦多大学Sheila McIlraith教授等

博弈树搜索

- 背景
- MiniMax算法
- Alpha Beta 剪枝
- 评价函数

搜索问题的泛化

- 至今我们考虑的搜索问题都假设智能体对环境有完全的控制
  - 除非智能体做出改变环境的行为，否则状态不会改变
- 这种假设并不总是合理的
  - 可能存在利益与你的智能体相违背的其他智能体
- 在这种情况下，我们需要通过扩展搜索的视角（泛化）来处理不是由我们的智能体所控制的状态的变化

搜索问题的泛化

	经典搜索	对抗 / 博弈搜索
环　　境	单智能主体	多智能主体
搜索方式	启发式搜索	对抗式搜索
优　　化	用启发式方法可找到最优解	因时间受限而被迫执行近似解
评价函数	路径的代价估计	博弈策略和局势评估

博弈类智力游戏

<table>
<tr><td rowspan="8">桌上游戏</td><td rowspan="4">棋盘游戏</td><td>井字棋</td></tr>
<tr><td>西洋跳棋</td></tr>
<tr><td>国际/中国象棋</td></tr>
<tr><td>围棋</td></tr>
<tr><td rowspan="2">纸牌游戏</td><td>德州扑克</td></tr>
<tr><td>桥牌</td></tr>
<tr><td>骰子游戏</td><td>僵尸骰子</td></tr>
<tr><td>棋牌游戏</td><td>麻将</td></tr>
<tr><td>猜拳游戏</td><td></td><td>剪刀石头布</td></tr>
</table>

现实世界问题

- 囚徒困境（Prisoner's dilemma）

两个犯罪分子被分别监禁，无法沟通。每个囚徒只有二选一的机会：揭发对方并证明其犯罪，或者保持沉默。可能的选项如下：

1）若囚徒A和B彼此揭发对方，则每个囚徒被监禁2年；

2）若囚徒A揭发B、而B保持沉默，则A被释放而B被监禁3年，反之亦然；

3）若A和B都保持沉默，则他们仅被监禁1年。

囚徒困境源于1950年梅里尔·弗拉德（Merrill Flood）等人的相关困境理论。

后来艾伯特·塔克（Albert W. Tucker）将其命名为“囚徒困境”。

博弈问题的复杂性

有人估算，围棋可能的棋局数约为10^{170}，比宇宙中原子的数目还要多！

棋盘游戏都属于完美信息博弈，面对其空间和时间的复杂性，人工智能的工作就是在每个决策点上寻找胜率最大的路径。

但是，非完美信息博弈，例如纸牌游戏和棋牌游戏，不仅仅是空间和时间的复杂性，还依赖于博弈策略。

现实生活中非完美信息的博弈几乎无处不在。

博弈算法的历史

- 1912年，恩斯特·策梅罗（Ernst Zermelo）提出了最小最大算法（MiniMax algorithm）。
- 1949年，克劳德·香农（Claude Shannon）采用评价函数和选择性搜索方法，开发了国际象棋软件。
- 1956年，约翰·麦卡锡（John McCarthy）在最小最大算法的基础上，提出了Alpha-beta剪枝方法。
- 同年，即1956年，亚瑟·塞缪尔（Arthur Samuel）开发了西洋跳棋（Checkers）程序，其中通过自我对弈来学习自身的评价函数。
- 1958年，亚历克斯·伯恩斯坦（Alex Bernstein）等人在IBM 704上开发了史上第一款计算机国际象棋软件。

西洋跳棋

博弈算法的历史

- 1968年，艾伯特·索伯里斯特实现了世界上首款计算机围棋程序。
- 1990年，中山大学化学系教授陈志行编写了“手谈”。在1995至1998年期间，手谈在国际计算机围棋比赛中七次获得冠军。
- 1993年，贝恩德·布鲁格曼（Bernd Brügmann）第一次将蒙特卡罗仿真（Monte Carlo Simulation）用于计算机围棋。
- 1994年，乔纳森·薛弗尔（Jonathan Schaeffer）开发了西洋跳棋程序，与人类世界冠军打了个平局。2007年，他与合作者在《Science》上发表了“Checkers is Solved（西洋跳棋已破解）”的论文。
- 1997年，IBM深蓝战胜了国际象棋冠军加里·卡斯帕罗夫。
- 2008年，雷米·库仑（Rémi Coulom）开发的围棋软件——疯石，在受让8子的情况下战胜了日本职业围棋四段青葉熏。

博弈算法的历史

- 2009年，斯坦福大学的Wan Jing Loh发表了“AI Mahjong（AI麻将）”的论文。
- 2011年，尾島陽児（Yoji Ojima）开发的围棋软件Zen19D达到了KGS（Kiseido Go Server, KGS）五段的水平。
- 2013年，疯石（Crazy Stone）在受让四子的情况下，战胜了日本九段棋手石田芳夫。
- 2015年2月，谷歌DeepMind公司通过深度强化学习的方法来控制视频游戏机Atari，达到了人类玩家的水平。
- 2015年12月，DeepMind公司的计算机围棋AlphaGo打败了欧洲围棋冠军樊麾（Fan Hui）。
- 2016年3月，AlphaGo在韩国首尔战胜了韩国九段职业棋手李世乭。

博弈算法的历史

- 2016年12月29日至2017年1月5日，以Master为网名在中国著名的围棋网站上与世界顶级围棋选手进行了60局在线快棋赛，赢得60连胜。
- 2017年5月， AlphaGo战胜了当时世界围棋排名第一的职业棋手柯洁。
- 2017年10月， DeepMind公司推出了从零开始学习的围棋软件AlphaGo Zero。接着又推出了会下围棋、国际象棋和日本将棋的Alpha Zero。
- 2016年12月，在一对一无限注德州扑克（Heads-Up No-Limit Texas Hold'em）比赛中，阿尔伯塔大学开发的人工智能扑克DeepStack击败了11位职业扑克选手。
- 2017年1月，卡内基·梅隆大学的人工智能扑克Libratus，在一对一无限注德州扑克比赛中，战胜了4位人类顶级选手。
- 2019年4月，OpenAI公司的OpenAI Five，在TI9的Dota2五对五终极决赛上，以2:0击败了上一届TI8世界冠军团队OG。

博弈的主要特点

- 每个玩家都有他们自身的利益取向
- 每个玩家都会根据自身的利益来改变世界（状态）
- 难点：你如何行动取决于你认为对方会如何行动，而对方如何行动又取决于他们认为你会如何行动

博弈的特征

- 两个玩家
- 离散的：游戏的状态或决策可以映射为离散的值
- 有限的：游戏的状态或可以采取的行动的种类是有限的

博弈的特征

- 零和博弈：完全的竞争
  - 游戏的一方赢了，则另一方输掉了同等的数量
  - 有些游戏并不具备这个特征
- 确定性：没有不确定的因素
  - 没有骰子，没有随机抽取的扑克牌，没有抛硬币等
- 完美信息：任何层面的状态都是可观察的
  - 比如：没有隐藏的卡牌

博弈的示例：剪刀，石头，布

- 剪刀可以剪布，布可以包石头，石头可以砸剪刀
- 可以用矩阵表示：玩家1选择一行，玩家2选择一列
- 每一格表示各个玩家结算的分数（玩家1的分数/玩家2的分数）
- 1：赢了，0：平局，-1：输了
- 所以这个游戏是零和博弈

Player II

Player I	R石头	P布	S剪刀
R	0/0	-1/1	1/-1
P	1/-1	0/0	-1/1
S	-1/1	1/-1	0/0

两玩家零和博弈的扩展

- 剪刀石头布是简单的一次性（one shot）的博弈
  - 每一方只有一次动作
  - 在博弈论中：属于策略或范式博弈
- 许多博弈是有多步的
  - 轮流：玩家是交替行动的
  - 比如，象棋、跳棋等
  - 在博弈论中：属于扩展形式的博弈
- 我们专注于扩展形式的博弈
  - 扩展形式的博弈中才会出现需要计算的问题

两玩家零和博弈的定义

- 两个玩家 A（Max）和 B（Min）
- 状态集合 S（游戏状态的有限集合）
- 一个初始状态I $\in S$（游戏的起始状态）
- 终止位置T $\in S$（游戏的终止状态：游戏结束时的状态）
- 后继函数（一个接收状态为输入，返回通过某些动作可以到达的状态的函数）
- 效益（Utility）或收益（payoff）函数U或者V：T $\rightarrow R$（将终止位置映射到实数的函数，表示每个终止位置对玩家A有多有利和对玩家B有多不利）

两玩家零和博弈的直观介绍

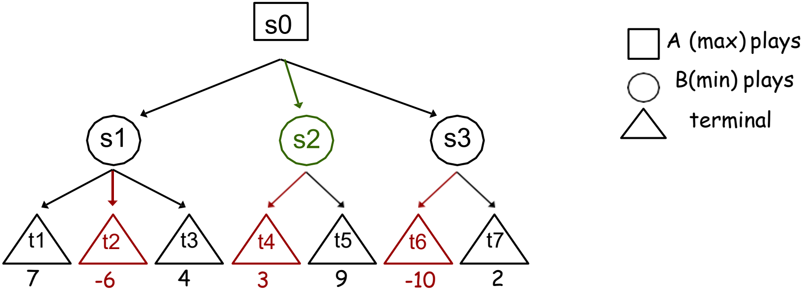
- 玩家交替行动（从玩家A，或玩家Max开始）
  - 当到达某个终止状态$t \in T$时游戏结束
- 一个游戏状态：一个（状态-玩家）对
  - 告诉当前是哪个状态，轮到哪个玩家行动
- 效益函数和终止状态代替原来的目标状态
  - 玩家A或Max希望最大化终止状态的效益
  - 玩家B或Min希望最小化终止状态的效益
- 另一种解读
  - 在终止状态t时，玩家A或Max获得了$U(t)$的收益，玩家B或Min获得了$-U(t)$的收益
  - 这就是为何称为“零和”

MiniMax算法

- 假设对方能总是做出最优的行动
  - 己方总是做出能最小化对方获得的收益的行动
  - 通过最小化对方的收益，可以最大化己方的收益

- 注意到如果已经知道在某些情况下对方无法做出最优的行动，那么可能存在比MiniMax更好的策略（也就是说，有其他的策略可以获得更多的收益）

MiniMax算法的收益



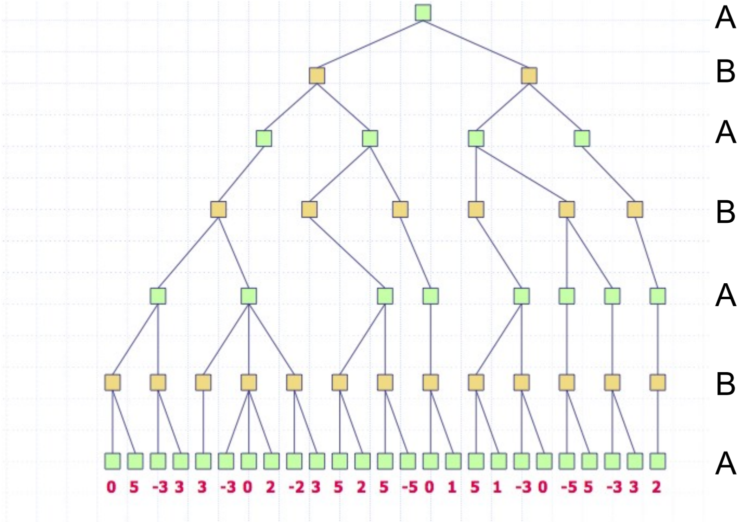
终止状态具有一个效益值（U或者V）。对于在非终止状态时的“效益值”，我们可以通过假设每个玩家都做出对自己最优的行动来计算得到。

MiniMax算法

- 构建完整的博弈树（每个叶子节点都表示终止状态）
  - 根节点表示起始状态，边表示可能的行动之类的
  - 每个叶子节点（终止状态）都标记了对应的效益值
- 反向传播效益值$U(n)$
  - 每个叶子节点t的$U(t)$值是预定义好的（算法输入的一部分）
  - 假如节点n是一个Min节点：
    - $U(n) = \min\{U(c): c$是n的子节点$\}$
  - 假如节点n是一个Max节点：
    - $U(n) = \max\{U(c): c$是n的子节点$\}$

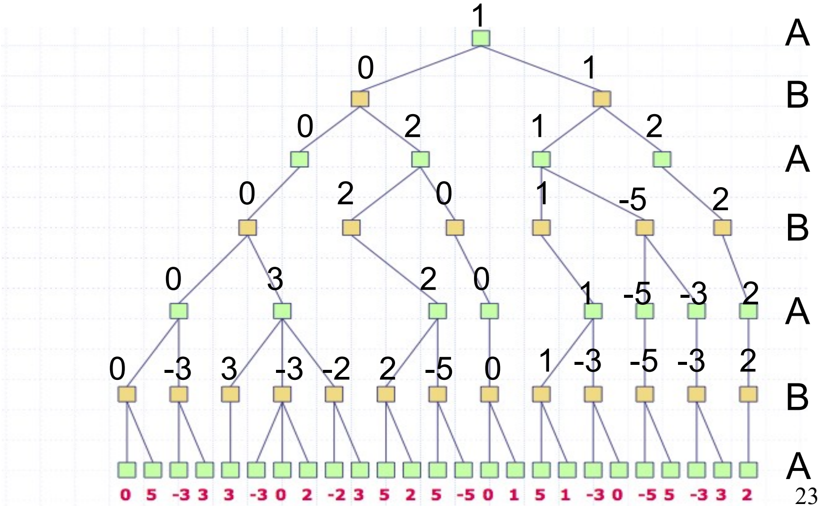
MiniMax算法

- 练习：计算出下面博弈树中的每个节点的理论效益值

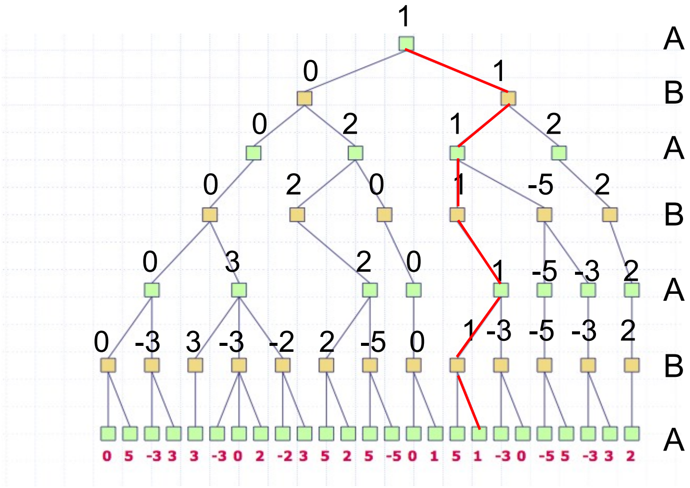


MiniMax算法

- 问题：假如每个玩家都按照对自己最优的策略行动，博弈树中的哪条路径会是游戏进行的过程?



MiniMax算法


1
A
0 1
B
0 2 1 2
A
0 2 0 1 -5 2
B
0 3 2 0 1 -5 -3 2
A
0 -3 3 -3 -2 2 -5 0 1 -3 -5 -3 2
B
A
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

MiniMax算法的深度优先实现

- 我们希望能构建整个博弈树并且记录每个玩家决策所需的值
- 但是博弈树的大小是指数增长的
- 之后我们会看到，其实知道整个博弈树并不必要
- 为了解决博弈树太大的问题，我们需要找到深度优先搜索算法来实现MiniMax
- 通过深度优先搜索我们可以找到MAX玩家的下一步动作（对于MIN玩家也是类似）
- 这样就可以避免记录指数级大小的博弈树，只需要计算我们需要的动作

MiniMax算法的深度优先实现

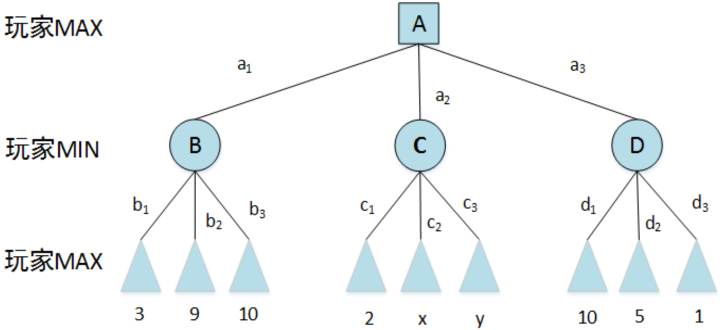
```
DFMiniMax(n, Player) //return Utility of state n given that
                     //Player is MIN or MAX

If n is TERMINAL
Return U(n) //Return terminal states utility
            //(U is specified as part of game)

//Apply Player's moves to get successor states.
ChildList = n.Successors(Player)
If Player == MIN
    return minimum of DFMiniMax(c, MAX) over c ∈ChildList
Else //Player is MAX
    return maximum of DFMiniMax(c, MIN) over c ∈ ChildList
```

- 这个算法要能运行，要求博弈树的深度是有限的
- 深度优先搜索的优点是：空间复杂度低

剪枝



$$\mathbf{minimax}(A) = \mathbf{max}(\mathbf{min}(3,9,10), \mathbf{min}(2,x,y), \mathbf{min}(10,5,1)) = \mathbf{max}(3, \mathbf{min}(2,x,y), 1)$$

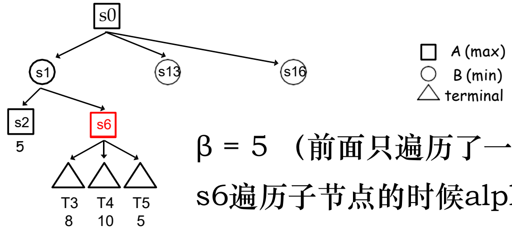
由于$\mathbf{min}(2,x,y)$的值小于等于2，所以即使不知道x和y的值，根结点A的效益值也可以算出来等于3。也就是说，MiniMax算法没有必要计算动作c_2和c_3对应的两个子树在游戏结束时所得效益值，就能决定在根结点采取动作a_1从而在游戏结束时可获得收益3，因此动作c_2和c_3所对应子树可以被剪枝。

剪枝

- MiniMax算法没有必要计算整棵博弈树
- 假设使用深度优先来生成博弈树
  - ✓ 只要计算节点n的一部分子节点就可以确定在MiniMax算法中我们不会考虑走到节点n了
  - ✓ 如果已经确定节点n不会被考虑，那么也就不用继续计算n的子节点了
- 有两种类型的剪枝：
  - ✓ 对Max节点的剪枝 (α-cuts)
  - ✓ 对Min节点的剪枝 (β-cuts)

对Max节点的剪枝 (α-cuts)

- 对于Max节点n:
- 设β是n被遍历过的兄弟节点中的最低效用值（n左边的兄弟节点已经被遍历过了）
- 设α是n被遍历过的子节点中的最高效用值（随着子节点的遍历而改变）



$\beta = 5$ （前面只遍历了一个兄弟节点）

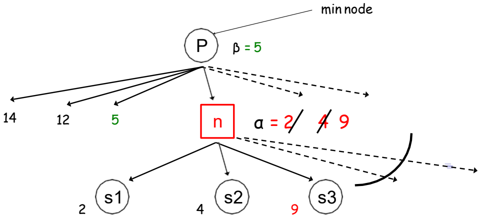
s6遍历子节点的时候alpha值的变化:

$\alpha = 8$; $\alpha = 10$; $\alpha = 10$

对Max节点的剪枝 (α-cuts)

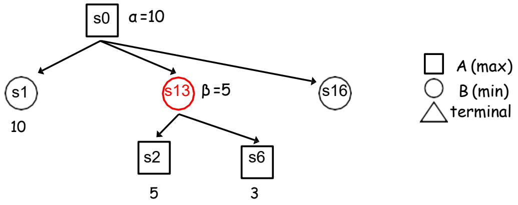
对于Max节点n，如果它的α值变得 ⩾ β的时候，就可以停止遍历n的子节点了

上层的Min节点不会来到n节点，因为Min节点一定会选择比n节点的效用值更小的兄弟节点



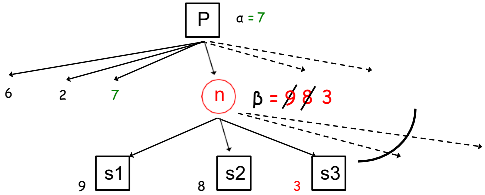
对Min节点的剪枝 (β-cuts)

- 对于Min节点n:
- 设α是到现在为止n节点的兄弟节点中最高的效用值 (在评估节点n时是固定的)
- 设β是到现在为止节点n的子节点中最低的效用值(changes as children examined)



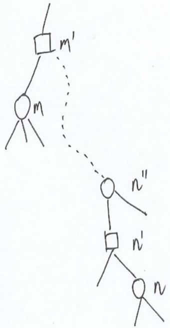
对Min节点的剪枝 (β-cuts)

- 如果β值变得 ≤ α，那么可以停止扩展n的子节点
- 上层的Max节点一定不会选择n节点，因为它会优先选择比n节点效用值更高的兄弟节点



- 泛化来说, 对于Min节点n, 如果β值变得 ≤ 某个Max祖先节点的α值, 那么就可以停止扩展n节点了

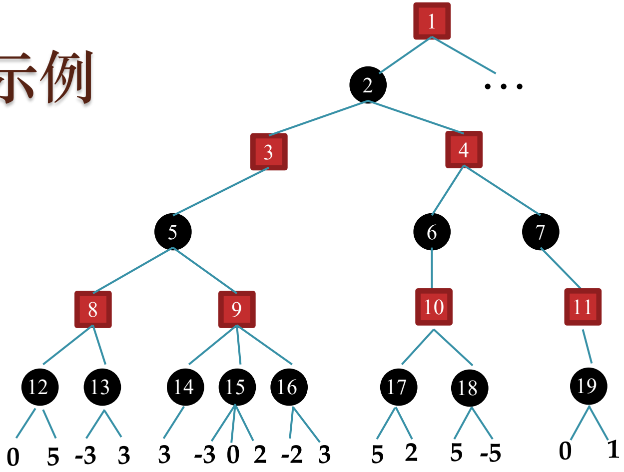
Alpha-beta剪枝的泛化



定理：如果α (m') = U (m) ⩾ β (n), 那么n节点可以被剪枝

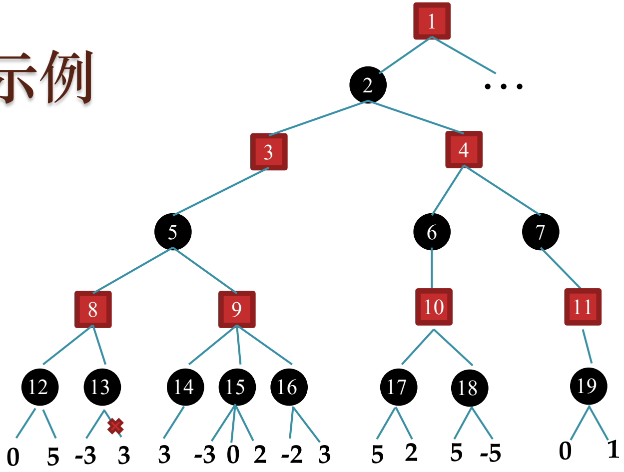
- 使用归纳法来证明
- Base case: m' = n'. 显然n节点可以被剪枝
- 接下来是归纳法的步骤：
- Case 1: α (n') > β (n). n的其它子节点不影响U (n')，所以n可以被剪枝
- Case 2: α (n') = β (n). 那么U (m) ⩾ β (n) = α (n') ⩾ β (n''). 根据归纳法，n''可以被剪枝，所以n也可以被剪枝

示例



记录Max节点的α值和Min节点的β值，它们分别表示Max方的最小得分和Min方的最大得分

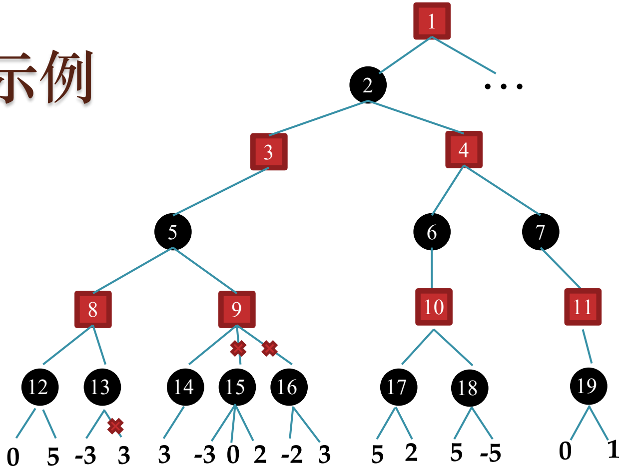
示例



$\beta(12) = 0$ => $U(12) = 0, \alpha(8) = 0$ => $\beta(13) = -3 <= \alpha(8)$ 对Min节点13执行β剪枝

记录Max节点的α值和Min节点的β值，它们分别表示Max方的最小得分和Min方的最大得分

示例

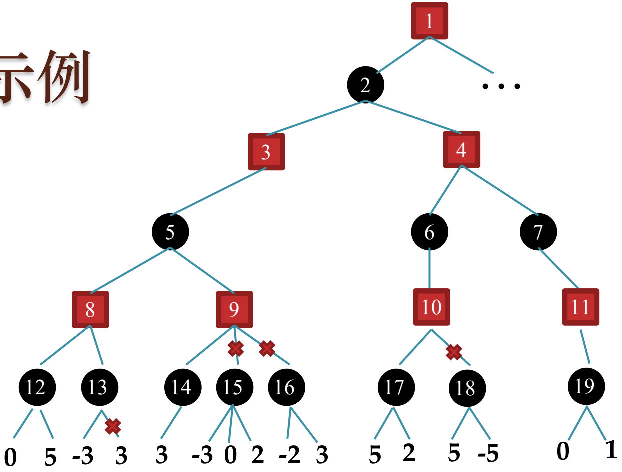


$\beta(12) = 0$ => $U(12) = 0, \alpha(8) = 0$ => $\beta(13) = -3 <= \alpha(8)$ 对Min节点13执行β剪枝

$\beta(5) = 0$ => $U(14) = 3, \alpha(9) = 3 >= \beta(5)$ 对Max节点9执行α剪枝

记录Max节点的α值和Min节点的β值，它们分别表示Max方的最小得分和Min方的最大得分

示例



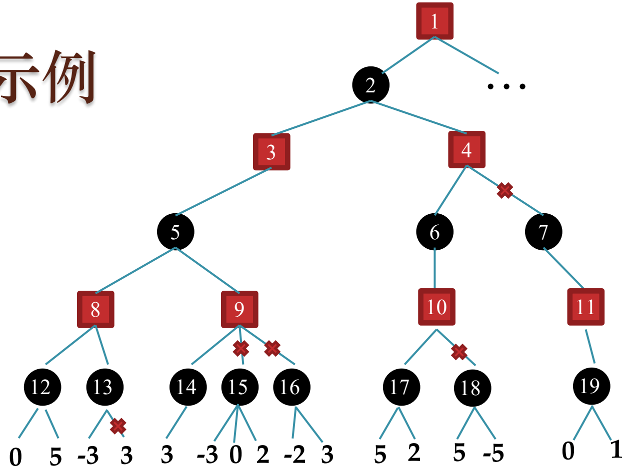
$\beta(12) = 0 \;\Rightarrow\; U(12) = 0, \alpha(8) = 0 \;\Rightarrow\; \beta(13) = -3 <= \alpha(8)$ 对Min节点13执行β剪枝

$\beta(5) = 0 \;\Rightarrow\; U(14) = 3, \alpha(9) = 3 >= \beta(5)$ 对Max节点9执行α剪枝

$\beta(2) = 0 \;\Rightarrow\; \beta(17) = 5, U(17) = 2, \alpha(10) = 2 >= \beta(2)$ 对Max节点10执行α剪枝

记录Max节点的α值和Min节点的β值，它们分别表示Max方的最小得分和Min方的最大得分

示例



$\beta(12) = 0$ => $U(12) = 0, \alpha(8) = 0$ => $\beta(13) = -3 <= \alpha(8)$ 对Min节点13执行β剪枝
$\beta(5) = 0$ => $U(14) = 3, \alpha(9) = 3 >= \beta(5)$ 对Max节点9执行α剪枝
$\beta(2) = 0$ => $\beta(17) = 5, U(17) = 2, \alpha(10) = 2 >= \beta(2)$ 对Max节点10执行α剪枝
$\alpha(4) = 2 >= \beta(2)$ 对Max节点4执行α剪枝

记录Max节点的α值和Min节点的β值，它们分别表示Max方的最小得分和Min方的最大得分

逐步运行Alpha-beta剪枝

- 在搜索过程中，记录Max节点的alpha值和Min节点的beta值，它们分别表示Max方的最小得分和Min方的最大得分
- Max节点的alpha值如果大于等于任何祖先Min节点的beta值，就进行alpha剪枝
- Min节点的beta值如果小于等于任何祖先Max节点的alpha值，就进行beta剪枝

实际操作中的问题

- 真实游戏很难生成整棵博弈树，例如，国际象棋的分支因子约为35，整棵博弈树会有2,700,000,000,000,000个节点。此时，即使使用alpha-beta剪枝也收效甚微，必须限制搜索树的深度。
- 真实游戏中根本无法扩展到叶子节点，因此需要启发式地计算非叶子节点的效用值，这样的启发式方法被称为评价函数。设计评价函数的基本要求如下：
  - ✓ 必须使得终止节点的排序和原来的效用函数一致；
  - ✓ 计算不能太耗时；
  - ✓ 对于非叶子节点，评价函数需要与这个节点实际能获得胜利的概率强相关。

如何设计评价函数

- 大多数的评价函数会分别计算各个特征的数值贡献，之后再进行结合
- 例如，在国际象棋游戏中，每个兵评价为1，马或象评价为3，车评价为5，皇后评价为9
- 数学上，一个加权评价函数为

$$Eval(s) = w_1 \cdot f_1(s) + \cdots + w_n \cdot f_n(s) = \sum_{i=1}^{n} w_i \cdot f_i(s)$$

- Deep Blue用了超过8000个特征
- 这里要考虑一个很强的假设：所有特征的贡献都是独立于其他特征的
- 这个假设不一定成立，因此有时也会用非线性的组合

如何设计评价函数

- 这些特征和权重都不是象棋规则的一部分
- 它们是由人类下棋的经验而来的
- 假如没有这方面的经验，则评价函数里的权重可以通过机器学习的技巧估计得到

Alpha-beta剪枝：井字棋

- 定义X_n为只有n个X而没有O的行，列或对角线的数量
- 同样O_n是只有n个O而没有X的行，列或对角线的数量
- 效用函数对于任何$X_3 = 1$的状态都赋值为+1
- 并且对于任何$O_3 = 1$的状态都赋值为-1
- 其他所有的终止状态效用都为0
- 对于非终止状态，我们使用线性评价函数

$$Eval(s) = 3X_2(s) + X_1(s) - (3O_2(s) + O_1(s))$$

Alpha-beta剪枝：井字棋

从空白棋盘开始建立博弈树，把树扩展到第2层

在博弈树第二层的每个节点上标记当前的评价函数值

使用MiniMax算法，标出第0和1层的节点的评价值

在最优节点最先生成的假设下，把第二层会被alpha-beta剪枝的节点圈出来

Alpha-beta剪枝：井字棋