



Game Theory Meets Computer Science

Part Two: Game Playing Algorithms

姜桂飞

南开大学 · 软件学院

G.Jiang@nankai.edu.cn

Exercise: Pick up a number

- Without showing your neighbor what you're doing, put in the box **below a whole number between 1 and 100**.
- We will calculate the average number chosen in the class.
- The **winner** in this game is the person whose number is closest to two-thirds times the average in the class.
- The winner will win one lucky coin and all other players win nothing.

Exercise: Pick up a number

Is it rational to play above $\frac{2}{3} \cdot 100 \approx 67$?

A: no (why?)

Given that other players are rational, is it rational to play above $\frac{2}{3} \cdot 67 \approx 44$?

A: no (same reasons)

Given that other players are rational, is it rational to play above $\frac{2}{3} \cdot 44 \approx 30$?

historical facts:

21.6 was the winning value in a large internet-based competition organized by the Danish newspaper [Politiken](#). This included 19,196 people and with a prize of 5000 Danish kroner.

Winner

- 参与人数: 106
- 平均数: 26.7
- 平均数的 $\frac{2}{3}$: 17.8

18

孔萍

Outline

- Background
- Game Tree
- Adversarial Search
 - Minimax Search
 - Alpha-Beta Pruning
- Monte-Carlo Tree Search



Background

Adversarial Search often Known as Games

Definitions of Game theory

- Study of strategic decision making . Specifically , study of mathematical models of conflict and cooperation between intelligent rational decision makers

Applications of Game theory

- Economics, political science, psychology, logic, computer science, and biology
- Behavioral relations and decision science, including both humans and non-humans (e.g. computers).

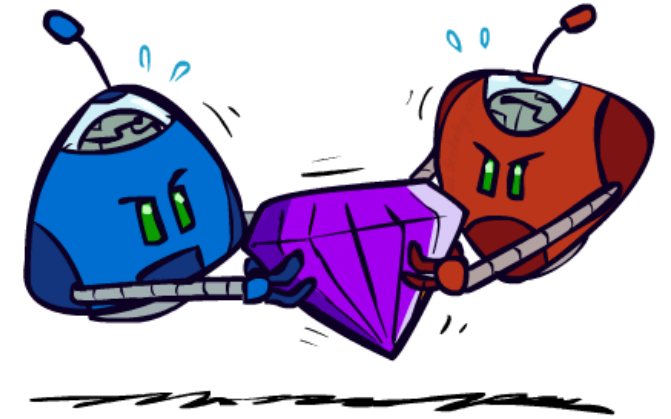
Features of games

- Two or more players (agents)
- Turn-taking vs. simultaneous moves
- Perfect information vs. imperfect information
- Deterministic vs. stochastic
- Cooperative vs. competitive
- Zero-sum vs. non zero-sum

Zero Sum vs. Non-zero Sum

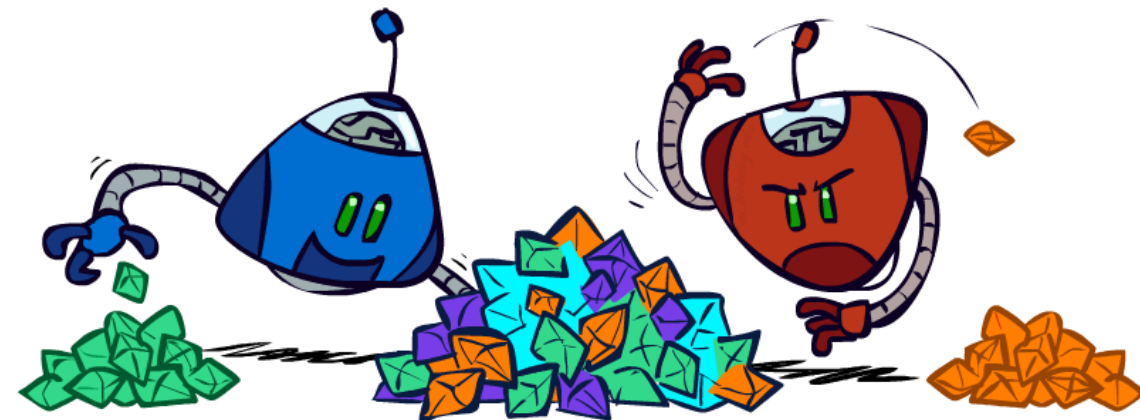
Zero sum games

- Agents have *opposite utilities*.
- Pure competition: win-lose, its sum is “zero”.



Non-zero sum games

- Agents have *independent utilities*.
- Cooperation, indifference, competition, ...
- Win-win, win-lose or lose-lose, its sum is not “zero”.



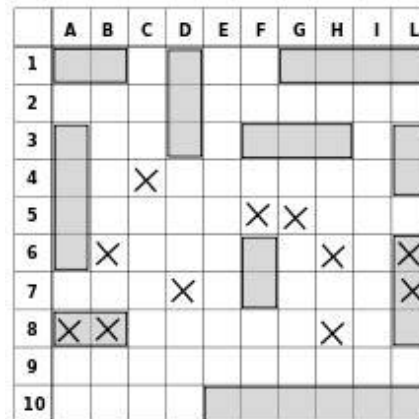
Types of Games

	Deterministic	Stochastic
Perfect information (fully observable)	Chess 国际象棋 Checkers 西洋跳棋 Go 围棋 Othello 黑白棋	Backgammon 西洋双陆棋 Monopoly 大富翁


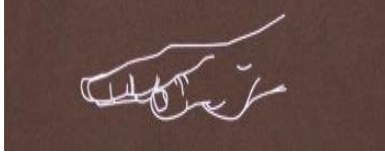






Types of Games

	Deterministic	Stochastic
Imperfect information (partially observable)	Stratego 西洋陆军棋 Battleships 海战棋	Bridge 桥牌 Poker 扑克 Scrabble 拼字游戏



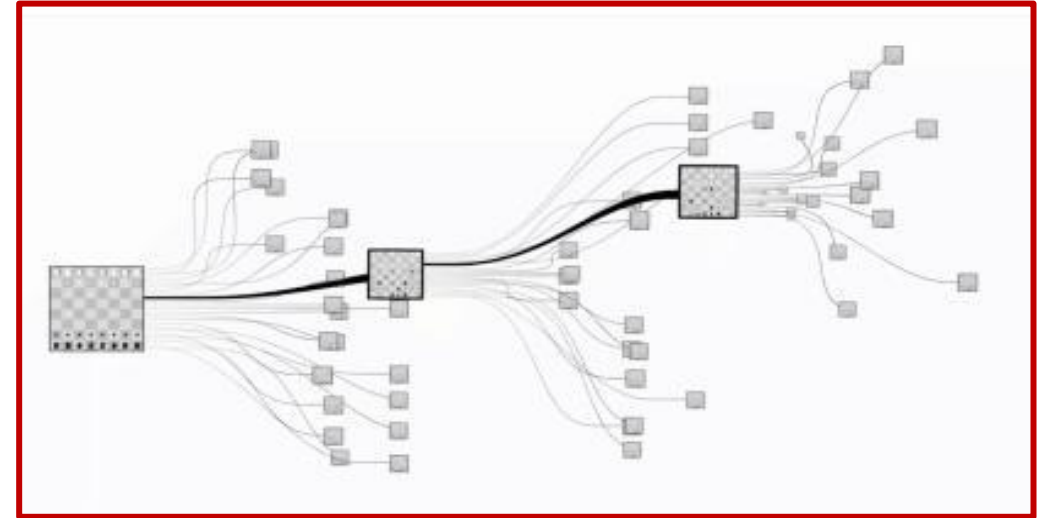
Ex: Rock-Paper-Scissor

		P2		
P1				
		0,0	-1,1	1,-1
		1,-1	0,0	-1, 1
		-1,1	1, -1	0,0

Which type?

Games are Interesting but Too Hard to Solve

- 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!



- Games, like the real world, therefore require the ability to make *some* decision even when calculating the *optimal decision* is infeasible.
- Game playing research has spawned a number of interesting ideas on how to make *the best possible use of time*

AI Algorithms



“Games are the perfect platform for developing and testing AI algorithms.”

-----Demis Hassabis

Origins of Game Playing Algorithms

1912	Ernst Zermelo 恩斯特·策梅洛	Minimax algorithm 最小最大算法
1949	Claude Shannon 克劳德·香农	Chess playing with evaluation function, selective search 用评价函数和选择性搜索下国际象棋
1956	John McCarthy 约翰·麦卡锡	Alpha-beta search Alpha-beta搜索
1956	Arthur Samuel 亚瑟·塞缪尔	Checkers program that learns its own evaluation function 学习自身的评价函数的西洋跳棋程序

- Ernst Zermelo(1871–1953), a German logician and mathematician.
- Claude Shannon (1916–2001), an American mathematician, and cryptographer known as “the father of information theory”.
- John McCarthy (1927-2011), an American computer scientist and cognitive scientist, and one of the founders of AI.
- Arthur Samuel (1901-1990), an American pioneer of computer gaming, AI, and ML.

Game Playing Algorithms Nowadays

Computers are better than humans

Checkers 西洋跳棋	Solved in 2007 2007 年已解决
Chess 国际象棋	IBM Deep Blue defeated Kasparov in 1997 IBM 深蓝于 1997 年战胜了卡斯帕罗夫
Go 围棋	Google AlphaGo beat Ke Jie in 2017 谷歌 AlphaGo 于 2017 年 3 月战胜了 柯洁

Computers are competitive with top human players

Backgammon 西洋双陆棋	TD Gammon used reinforcement learning to learn evaluation function TD Gammon 使用了强化学习方法来得到评价函数
Bridge 桥牌	Top systems use Monte Carlo simulation & alpha-beta search 顶级的系统使用 蒙特卡罗仿真和 alpha-beta 搜索



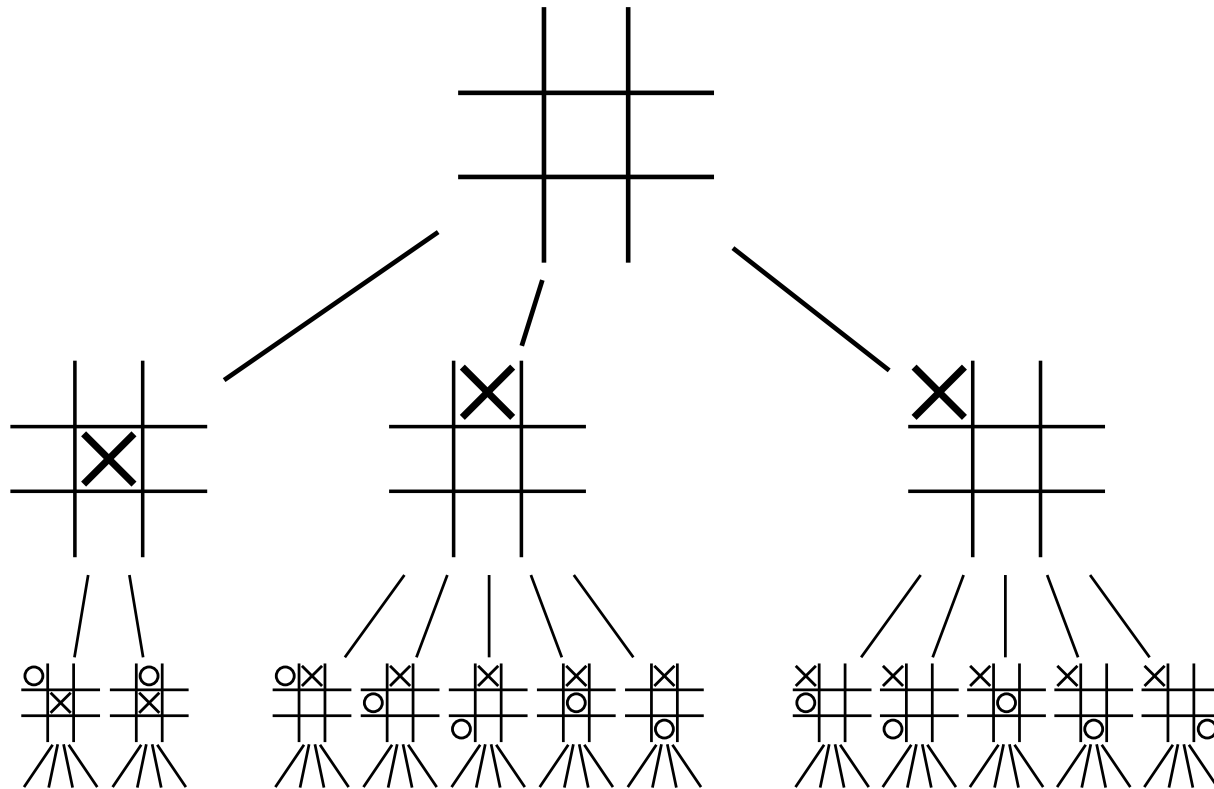
Game Tree

Two Players Games

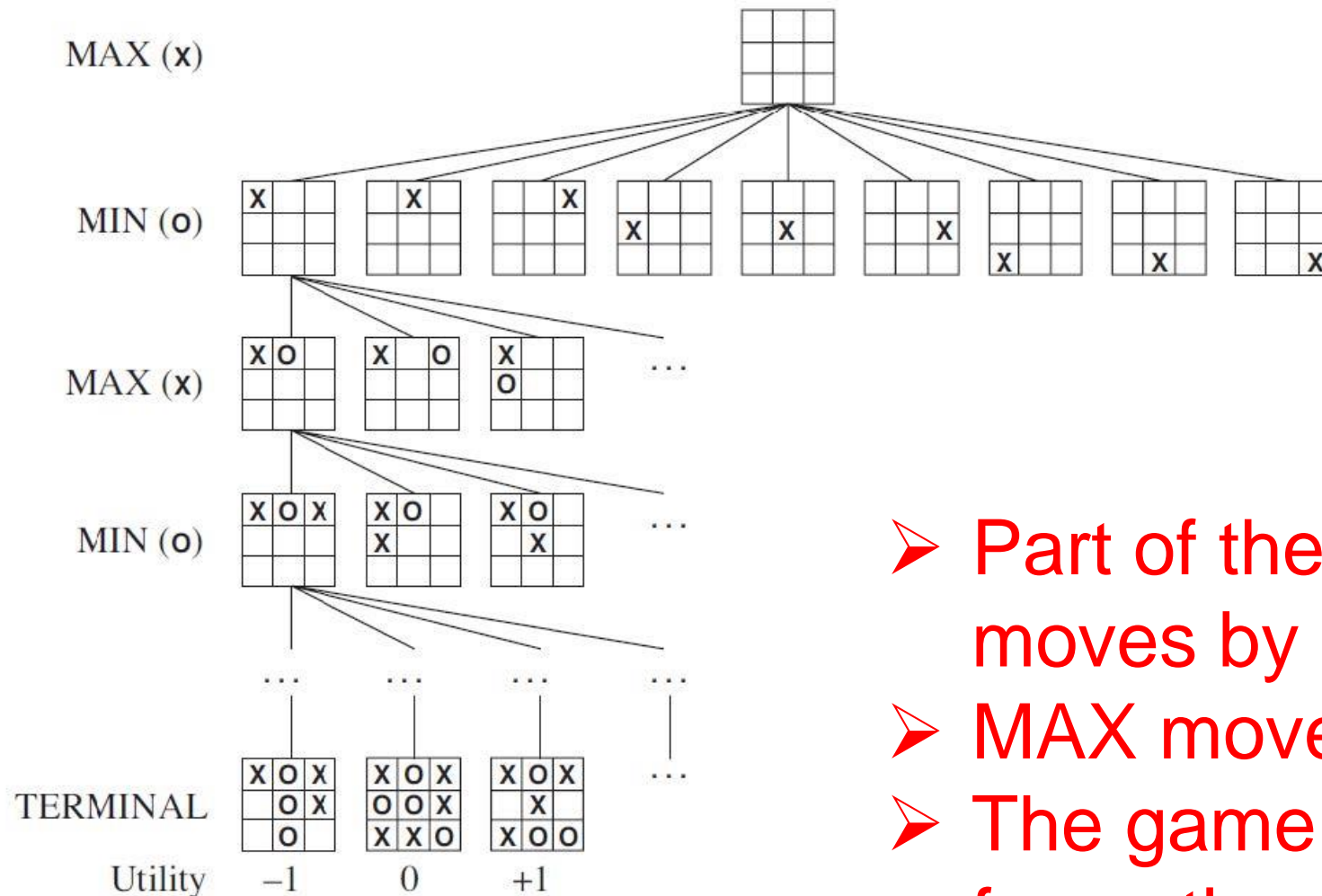
- **Feature**
 - deterministic, perfect information, turn-taking, two players, zero-sum.
- **Calling the two players**
 - MAX, MIN.
 - MAX moves first, and then they take turns moving, until the game is over.
- **At game end**
 - winner: award points
 - loser: give penalties.

Game Tree

- A directed graph whose nodes are positions in a game and whose edges are moves.

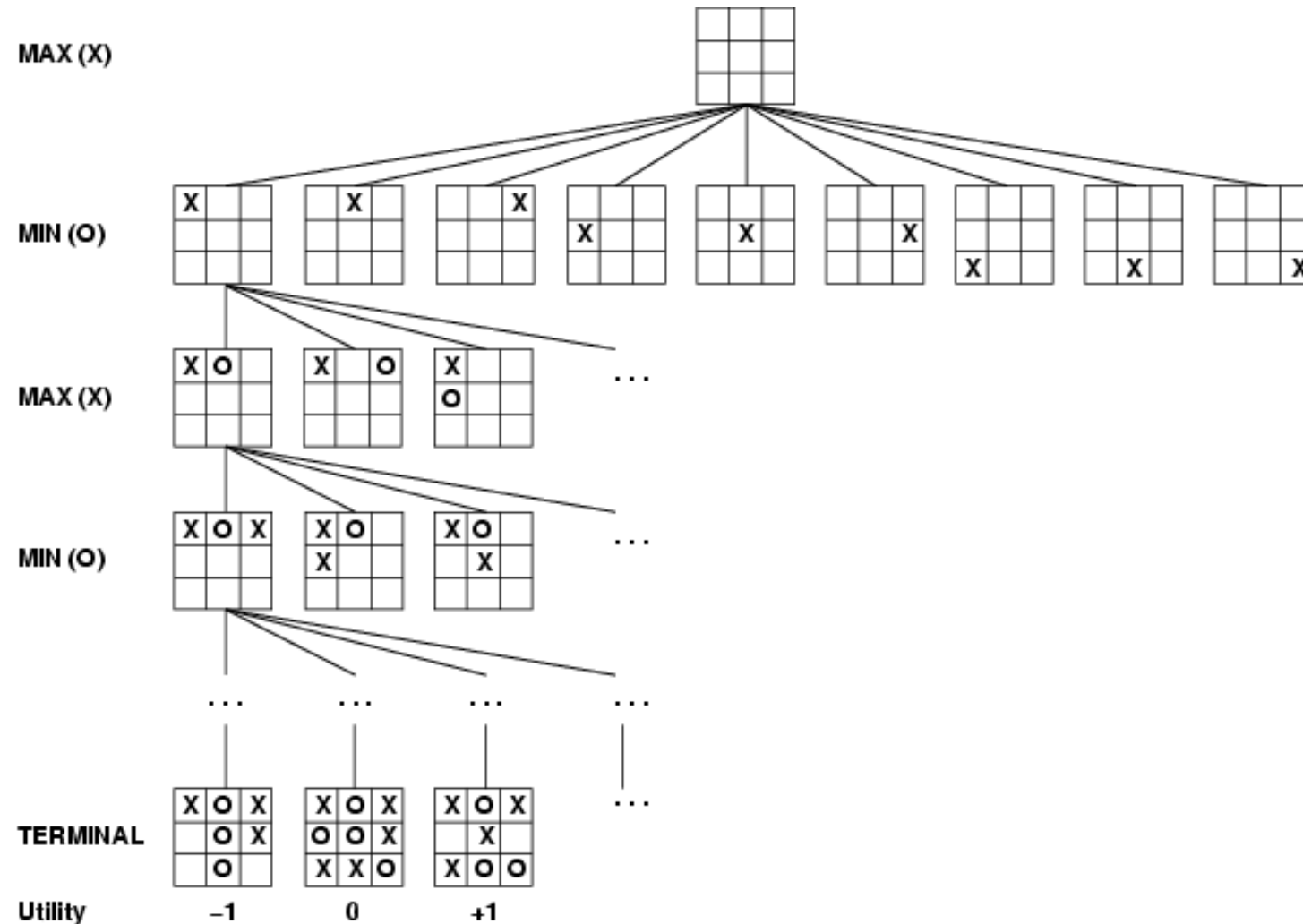


Example: Game Tree of Tic-tac-toe



- Part of the tree, giving alternating moves by MIN(O) and MAX(X).
- MAX moves first.
- The game tree is relatively small, fewer than $9! = 362,880$ nodes.

Example: Game Tree of Tic-tac-toe



How do we search this tree to find the optimal move?

Search versus Games

- Search – no adversary
 - Solution is (heuristic) method for finding goal
 - Heuristics and CSP techniques can find *optimal* solution
 - Evaluation function: estimate of cost from start to goal through given node
 - Examples: path planning, scheduling activities
- Games – adversary
 - Solution is strategy
 - strategy specifies move for every possible opponent reply.
 - Time limits force an *approximate* solution
 - Evaluation function: evaluate “goodness” of game position
 - Examples: chess, checkers, Othello, backgammon

Games as Search

- Two players: MAX and MIN
- MAX moves first and they take turns until the game is over
 - Winner gets reward, loser gets penalty.
 - “Zero sum” means the sum of the reward and the penalty is a constant.
- MAX uses search tree to determine next move.

Formal definition as a search problem

- **Initial state:** Set-up specified by the rules
- **Player(s):** Defines which player has the move in a state.
- **Actions(s):** Returns the set of legal moves in a state.
- **Result(s,a):** Transition model defines the result of a move.
- **Terminal-Test(s):** Is the game finished? True if finished, false otherwise.
- **Utility function(s,p):** Gives numerical value of terminal state s for player p .
 - E.g., win (+1), lose (-1), and draw (0) in tic-tac-toe.
 - E.g., win (+1), lose (0), and draw (1/2) in chess.



Minimax Search

Optimal Solution

In normal search

- The optimal solution would be a sequence of actions leading to a goal state (terminal state) that is a win.

In adversarial search

- Both of MAX and MIN could have an optimal strategy.
- In initial state, MAX must find a strategy to specify MAX's move
- then MAX's moves in the states resulting from every possible response by MIN, and so on.

Minimax Theorem

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

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

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*.

Optimal Solution in Adversarial Search

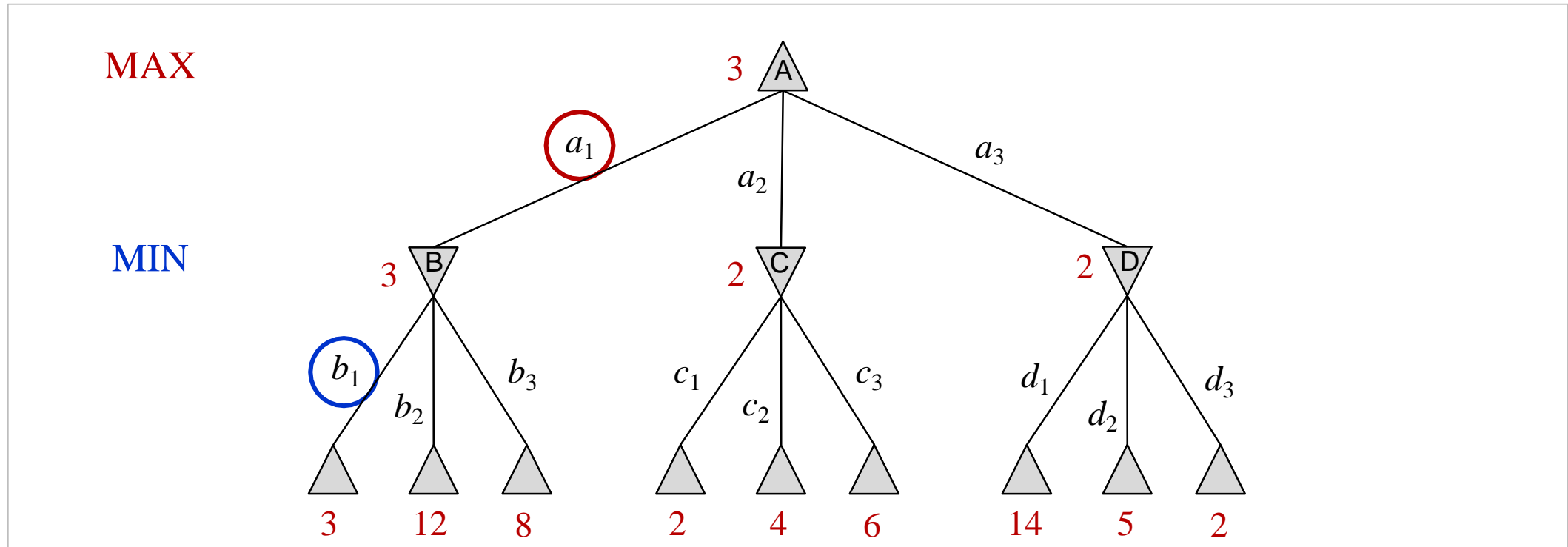
- Given a game tree, **the optimal strategy** can be determined from the minimax value of each node, write as MINIMAX(n).
- Assume that both players play optimally from there to the end of the game.

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

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

Minimax Decision -- A Two-player Game Tree



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

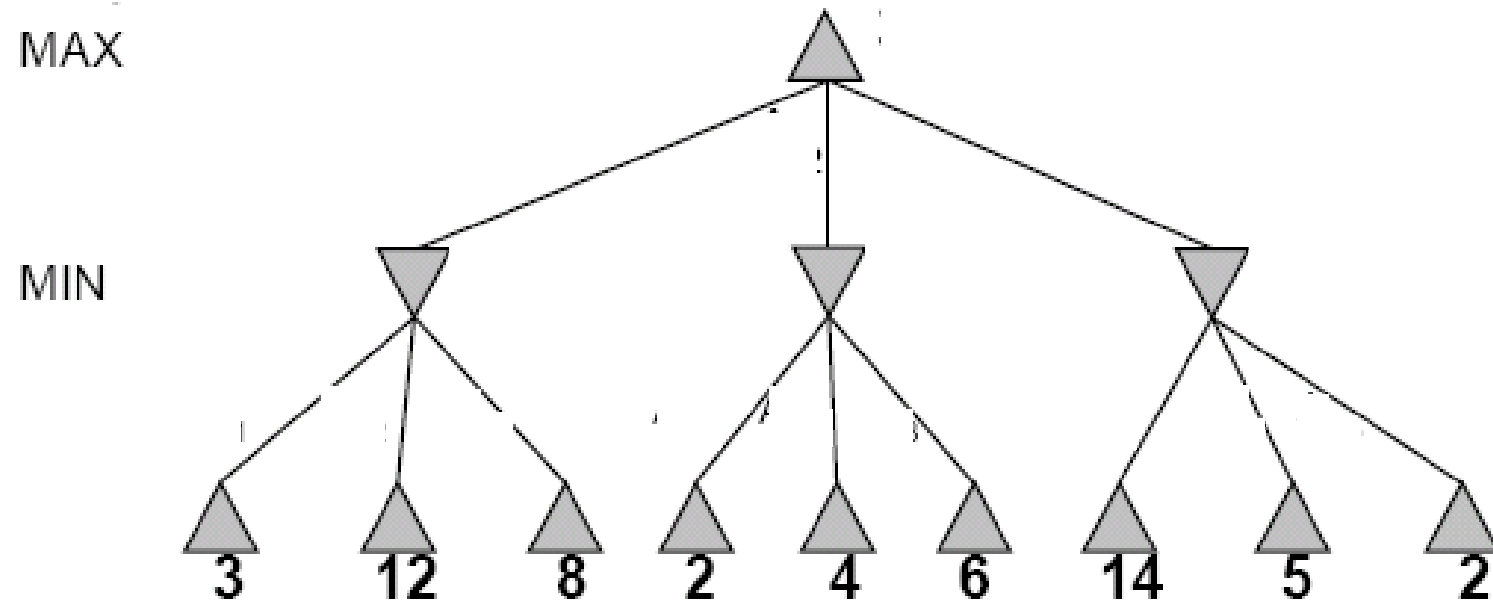
MIN's best reply at B is b_1 (with the **lowest** minimax value)

Minimax Algorithm

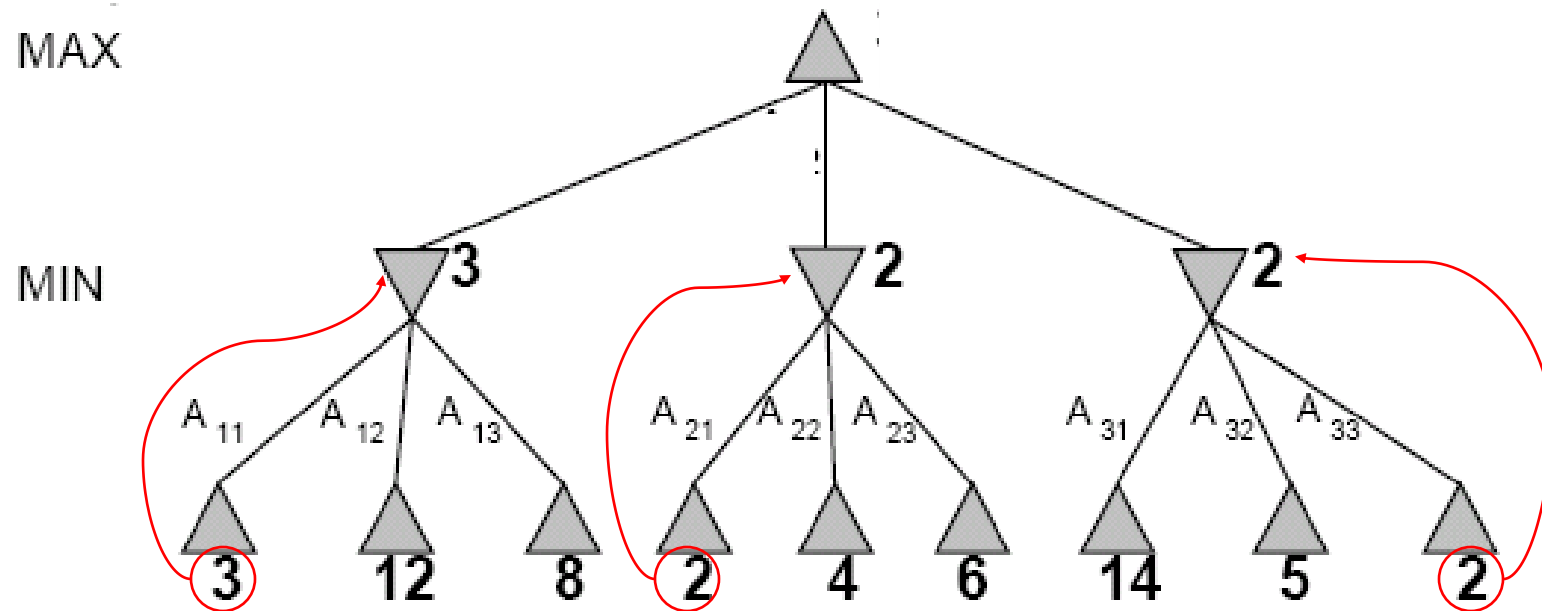
Designed to find **the optimal strategy for Max** and find best move:

- 1. Generate the whole game tree, down to the leaves.
- 2. Apply utility (payoff) function to each leaf.
- 3. Back-up values from leaves through branch nodes:
 - a Max node computes the Max of its child values
 - a Min node computes the Min of its child values
- 4. At root: choose the move leading to the child of highest value.

A Two-player Game Tree

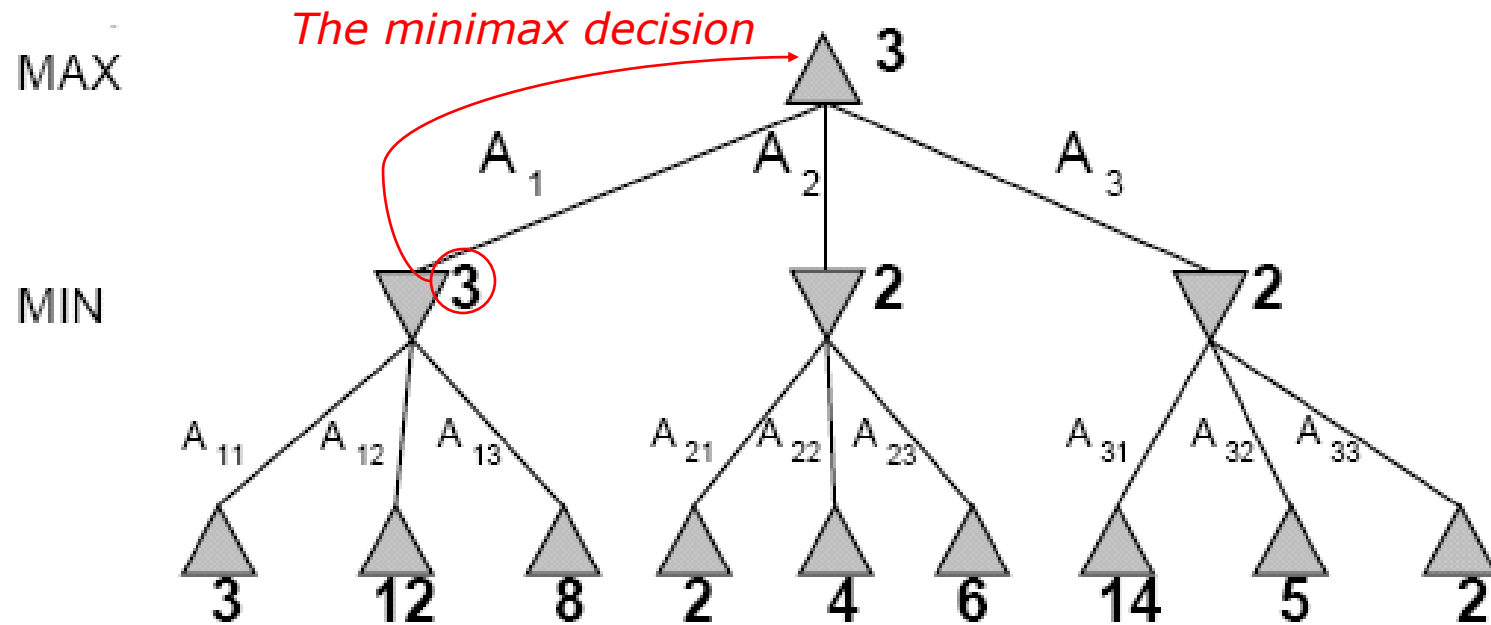


A Two-player Game Tree



A Two-player Game Tree

Minimax maximizes the utility for the worst-case outcome for max



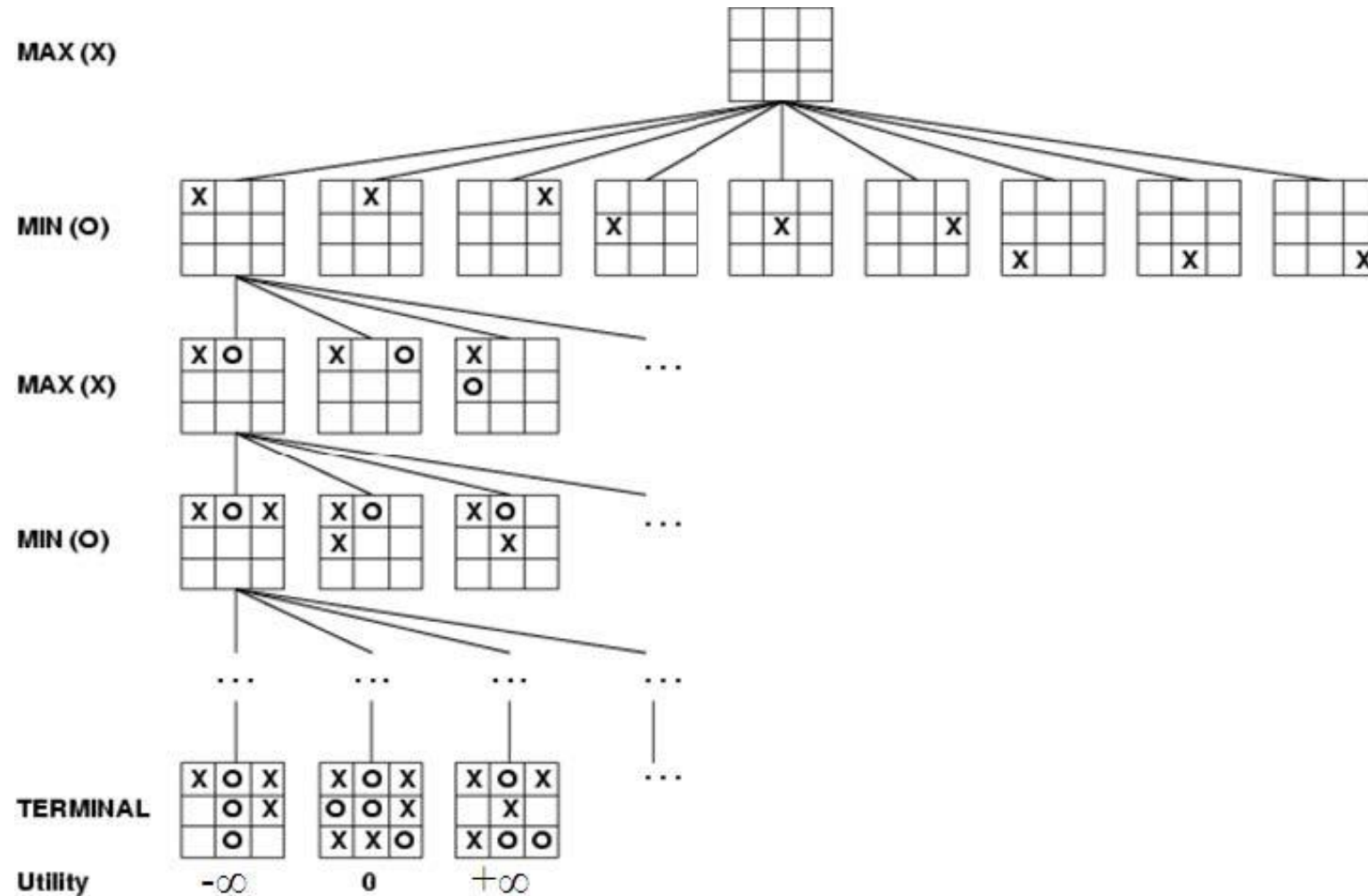
Pseudocode for Minimax Algorithm

function MINIMAX-DECISION(*state*) **returns** an action
 return $\operatorname{argmax}_{a \in \text{ACTIONS}(s)} \text{MIN-VALUE}(\text{RESULT}(\text{state}, a))$

function MAX-VALUE(*state*) **returns** a utility value
 if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
 $v \leftarrow -\infty$
 for each *a* **in** ACTIONS(*state*) **do** $v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(\text{RESULT}(\text{state}, a)))$
 return *v*

function MIN-VALUE(*state*) **returns** a utility value
 if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
 $v \leftarrow +\infty$
 for each *a* **in** ACTIONS(*state*) **do** $v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(\text{RESULT}(\text{state}, a)))$
 return *v*

Ex: Tic-Tac-Toe



Applying
MiniMax to tic-
tac-toe

Problem with minimax search

Problem with minimax search

- Minimax search needs to generate the whole game tree
- Number of game states is exponential in depth of the tree.

E.g. Chess

$b \approx 35$ (approximate average branching factor)

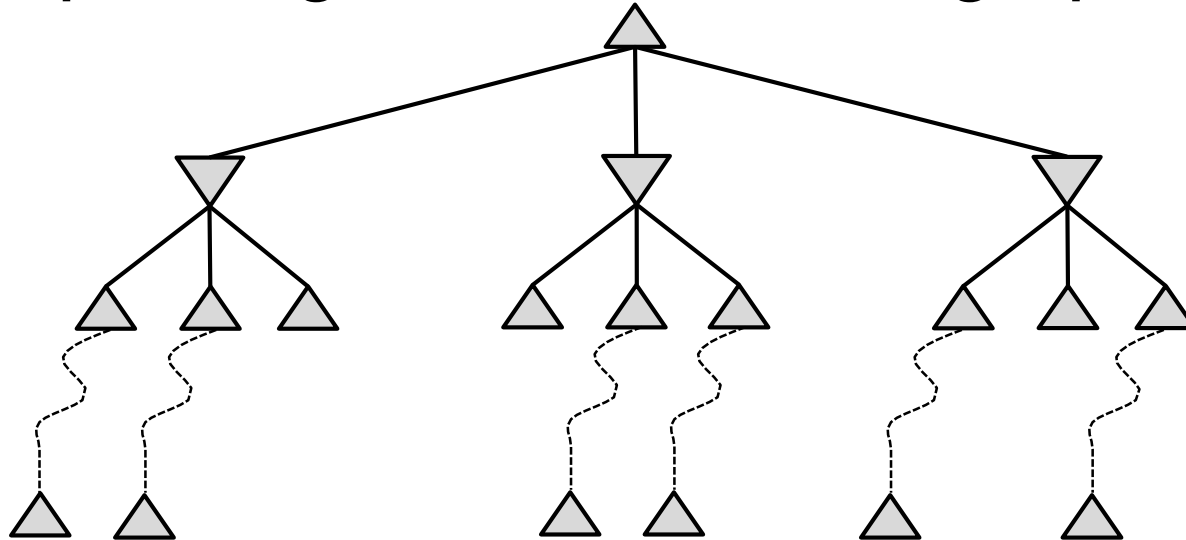
$d \approx 100$ (depth of game tree for “typical” game)

$b^d \approx 35^{100} \approx 10^{154}$ nodes!!

It is usually impossible to develop the whole search tree

One Solution

- Compute correct minimax decision without looking at every node in game tree.
- That is, use “pruning” to eliminate large parts of the tree.



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)

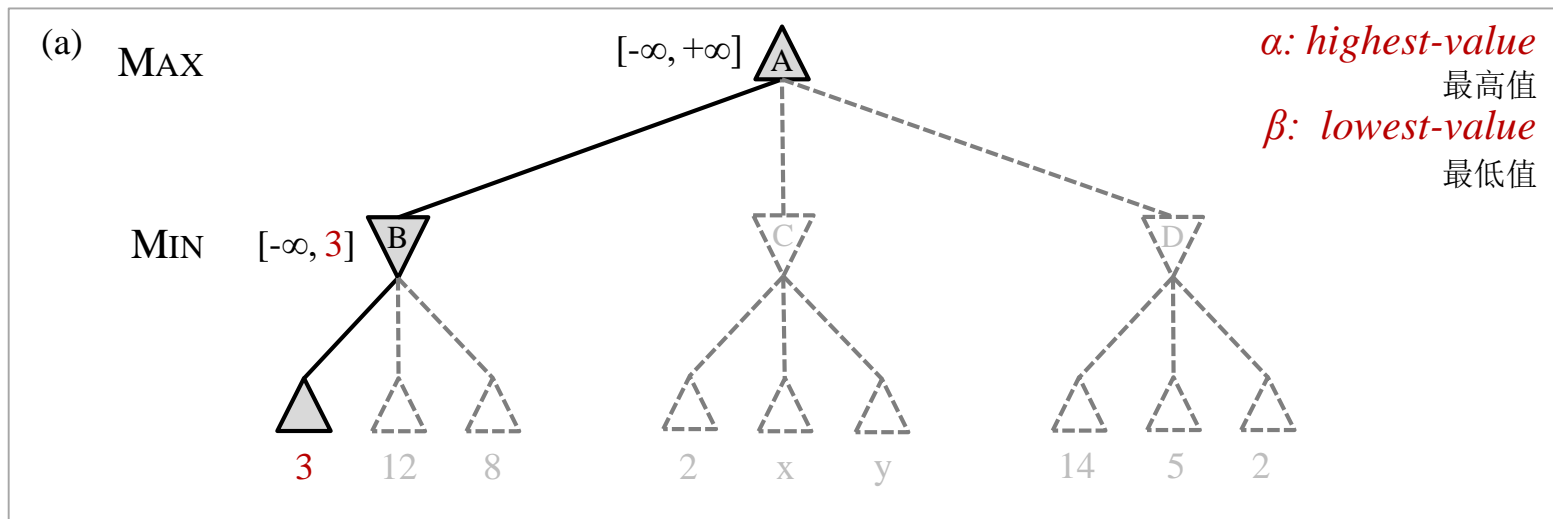


Alpha-Beta Pruning

Alpha-Beta Pruning

- A search algorithm to decrease the number of nodes that are evaluated by the minimax algorithm.
 - α : highest-value we have found so far at any point along the path for MAX.
 - β : lowest-value we have found so far at any point along the path for MIN.
- Alpha–beta search respectively:
 - updates the values of α and β as it goes along, and
 - 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.

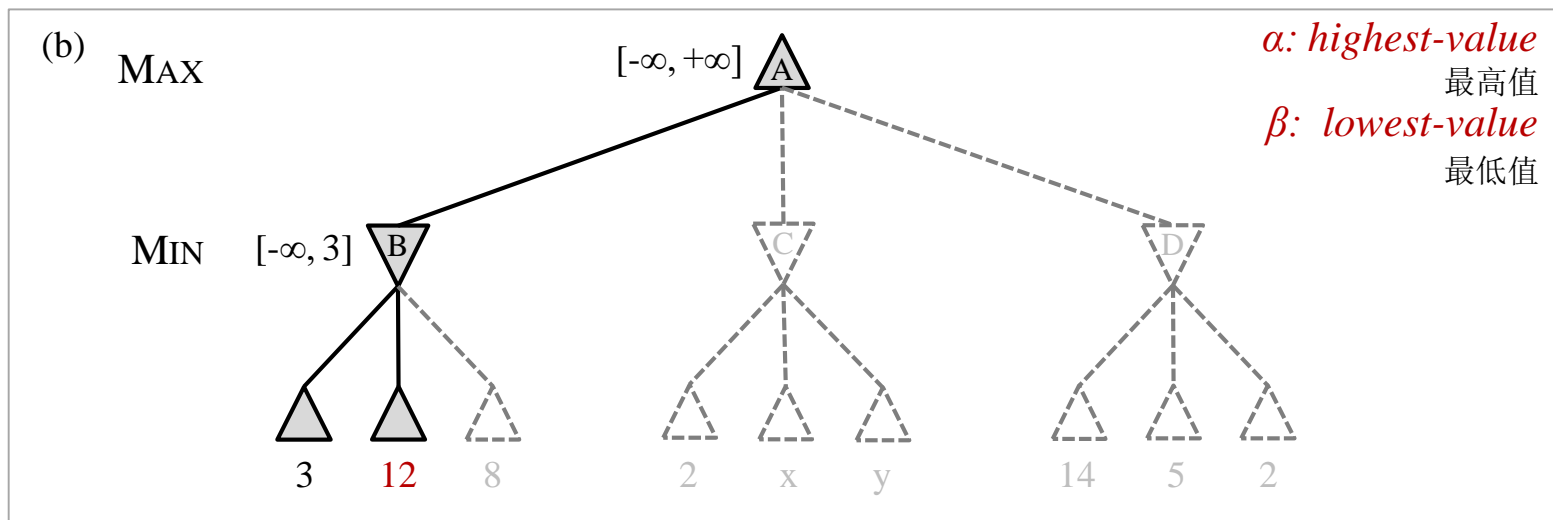
Example: Game Tree Using Alpha-Beta Pruning



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, $B[\beta=3]$.

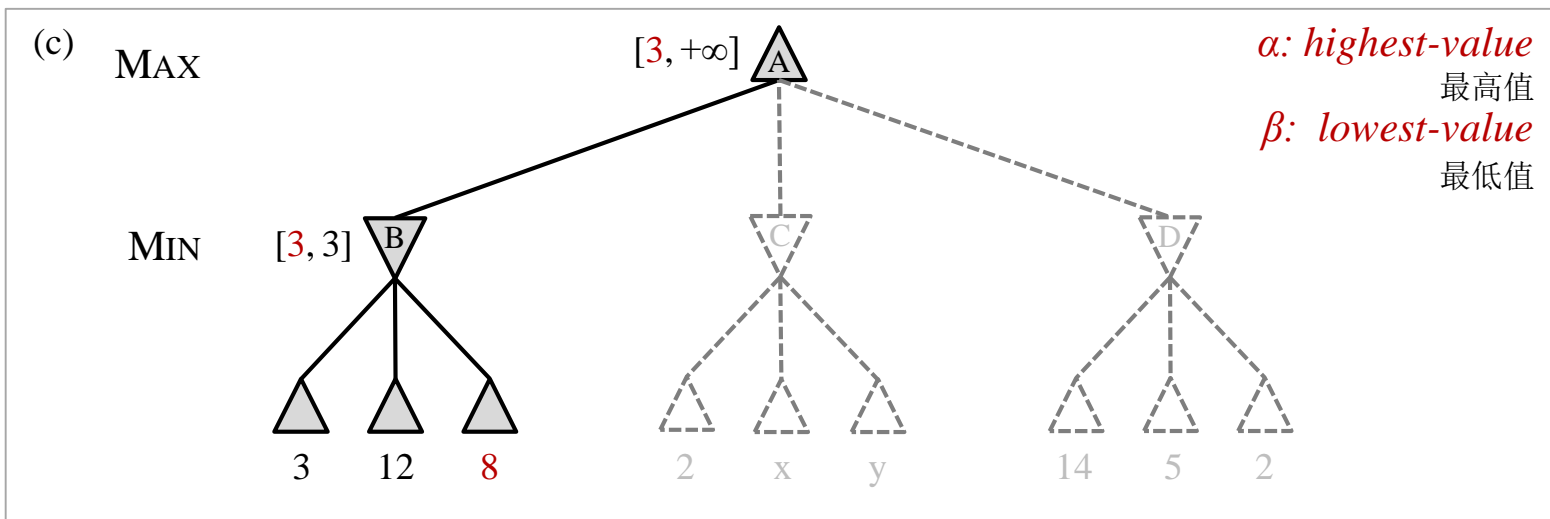
B 下面第一个叶节点的值是3。因此， B 作为MIN节点， $B[\beta=3]$ 。



(b) The 2nd leaf below B has a value of 12; MIN would avoid this move, still $B[\beta=3]$.

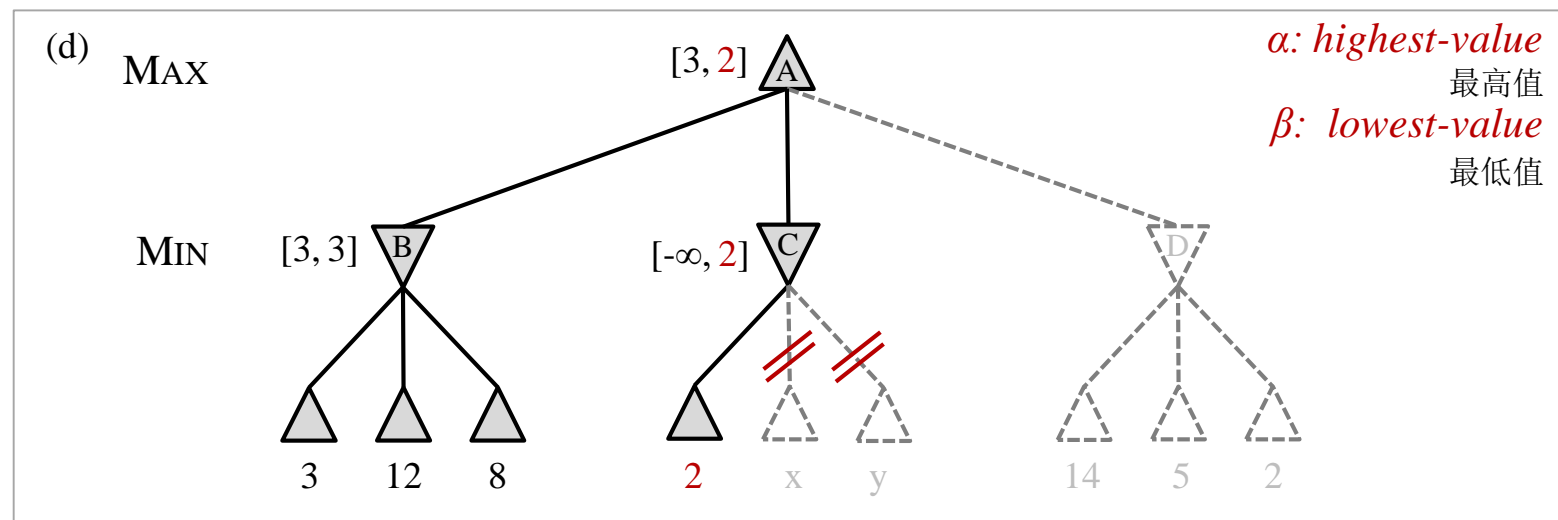
B 下面第二个叶节点的值是12；MIN将回避这个移动，仍然是 $B[\beta=3]$ 。

Example: Game Tree Using Alpha-Beta Pruning



(c) The 3rd leaf below B has a value of 8; so exactly MIN node $B[\beta=3]$. Now, we can infer $B[\alpha=3]$, because MAX has $A[\alpha \geq 3]$.

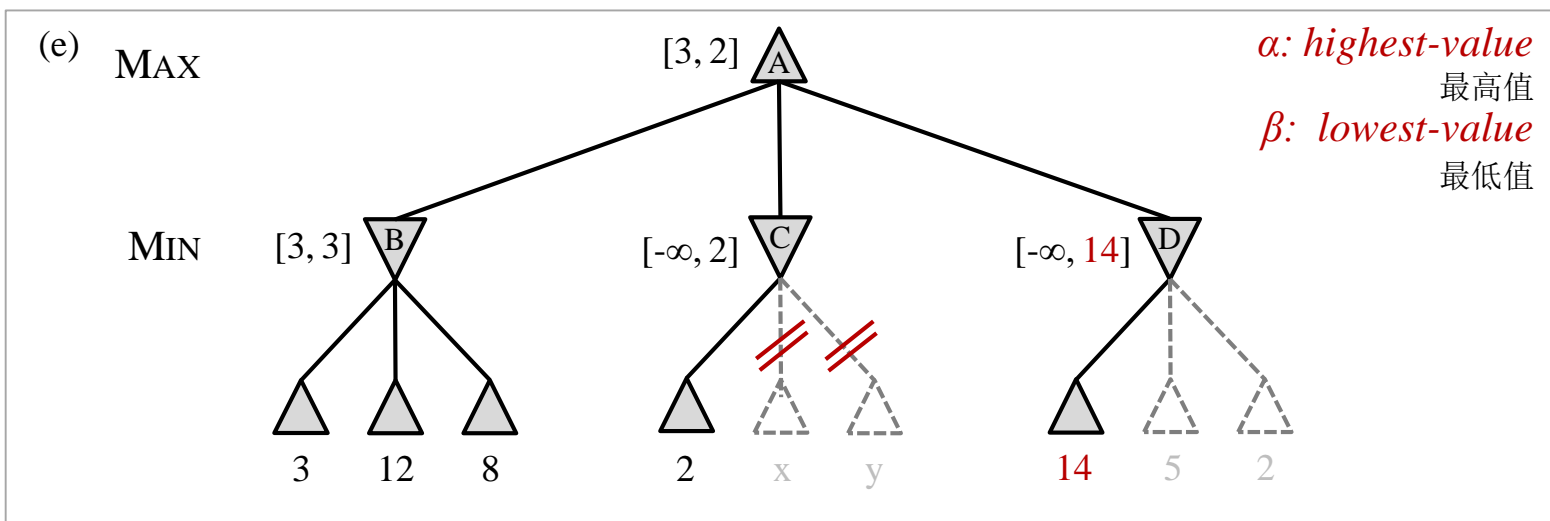
B 下面第三个叶节点的值是8；故MIN节点正是 $B[\beta=3]$ 。现在，因为MAX为 $A[\alpha \geq 3]$ ，我们能够推出 $B[\alpha=3]$ 。



(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).

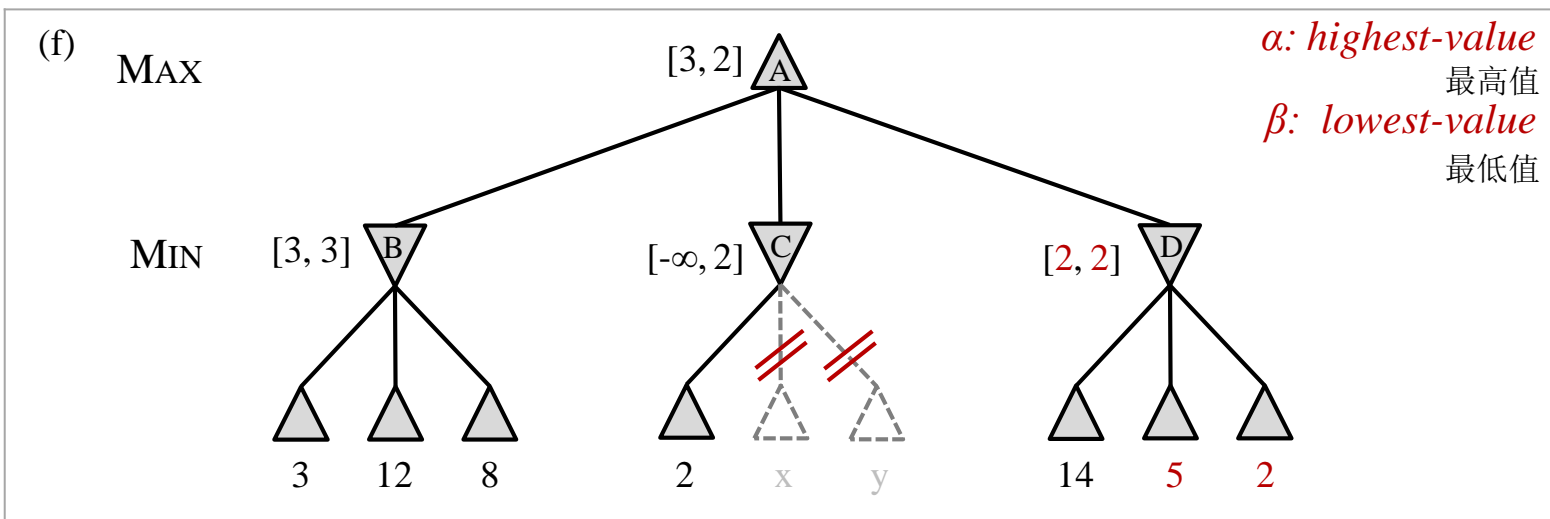
C 下面第一个叶节点的值是2，因此，由于MIN节点 $C[\beta=2]$ ，且 $B[\beta=3] > C[\beta=2]$ ，故MAX将不会选择 C ，所以只需剪掉 C 的所有后继节点 (α - β pruning)。

Example: Game Tree Using Alpha-Beta Pruning



(e) The 1st leaf below D is 14, $D[\beta \leq 14]$, so we need to keep exploring D 's successor states. We now have bounds on all of root's successors, so $A[\beta \leq 2]$.

D 下面第一个叶节点为14, $D[\beta \leq 14]$, 故我们需要不断搜索 D 节点的后继状态。到此我们已经遍历了根节点的所有后继节点, 故 $A[\beta \leq 2]$ 。

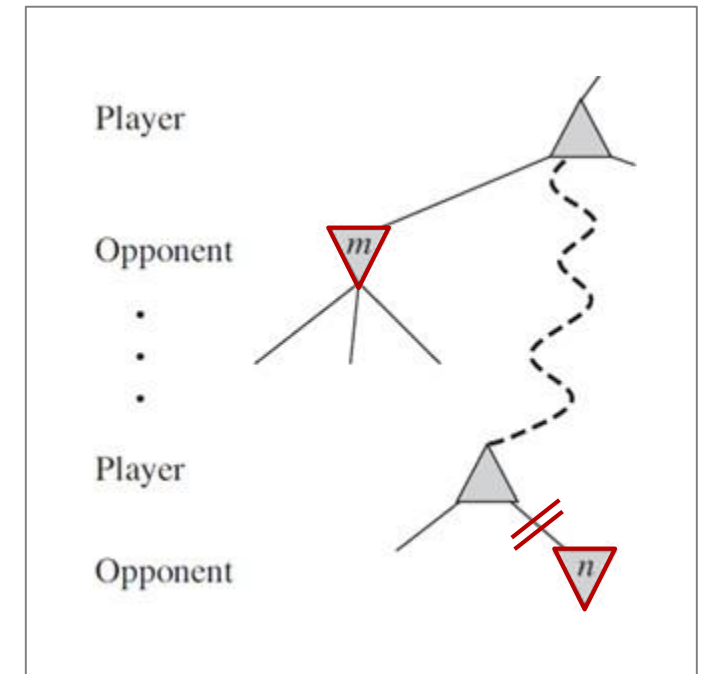


(f) The 2nd successor of D is worth 5, so keep exploring. The 3rd successor is worth 2, so $D[\beta = 2]$. MAX's decision at the root keeps $A[\beta = 2]$.

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

- Alpha-beta pruning can be applied to trees of any depth, and often possible to **prune entire subtrees** rather than just leaves.
- The general principle:
 - Consider a node n somewhere in the tree, such that Player has a choice of moving to that node.
 - 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.



Alpha-Beta Algorithm

- Depth first search
 - only considers nodes along a single path from root at any time
- α = highest-value choice found at any choice point of path for MAX
(initially, $\alpha = -\text{infinity}$)
- β = lowest-value choice found at any choice point of path for MIN
(initially, $\beta = +\text{infinity}$)
- Pass current values of α and β down to child nodes during search.
- Update values of α and β during search:
 - MAX updates α at MAX nodes
 - MIN updates β at MIN nodes
- Prune remaining branches at a node when $\alpha \geq \beta$

When to Prune

- Prune whenever $\alpha \geq \beta$.
 - Prune below a Max node whose alpha value becomes greater than or equal to the beta value of its ancestors.
 - **Max nodes update alpha** based on children's returned values.
 - Prune below a Min node whose beta value becomes less than or equal to the alpha value of its ancestors.
 - **Min nodes update beta** based on children's returned values.

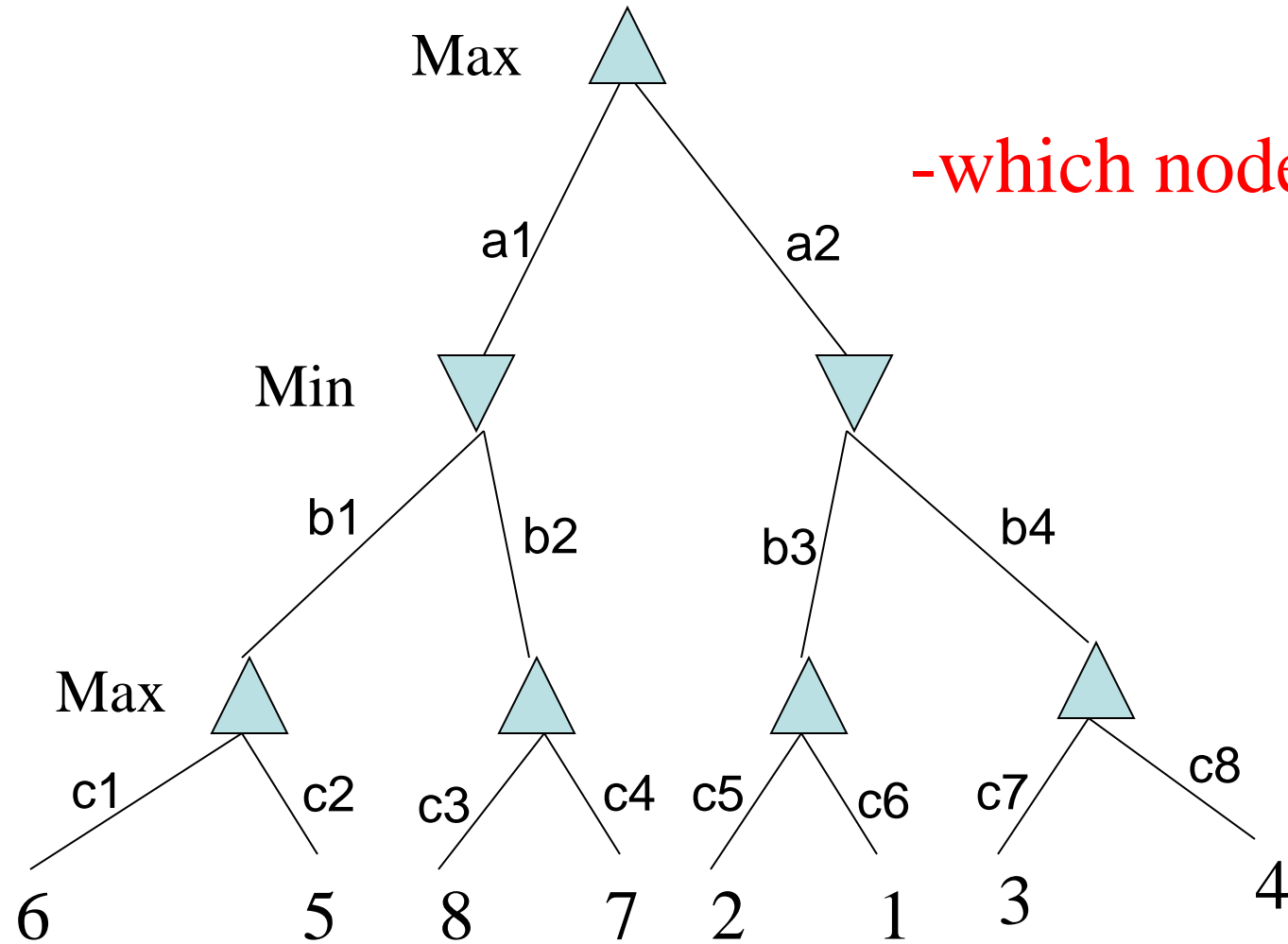
Alpha-Beta Search Algorithm

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

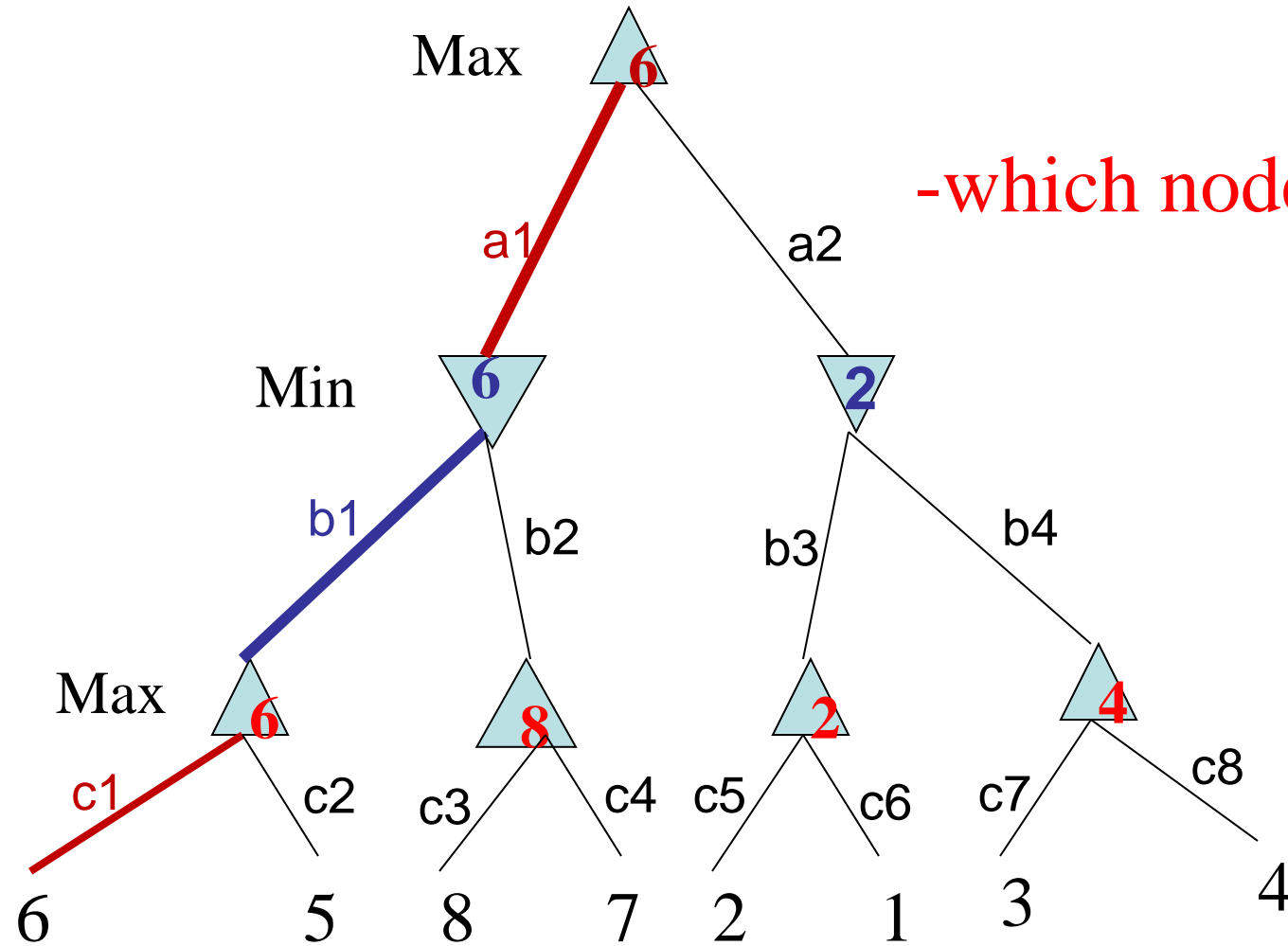
function MAX-VALUE(*state*, α , β) **returns** a utility value
 if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
 $v \leftarrow -\infty$
 for each a **in** ACTIONS(*state*) **do**
 $v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(\text{RESULT}(\text{state}, a), \alpha, \beta))$
 if $v \geq \beta$ **then return** v **else** $\alpha \leftarrow \text{MAX}(\alpha, v)$
 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**
 $v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(\text{RESULT}(\text{state}, a), \alpha, \beta))$
 if $v \geq \alpha$ **then return** v **else** $\beta \leftarrow \text{MIN}(\beta, v)$
 return v

Example

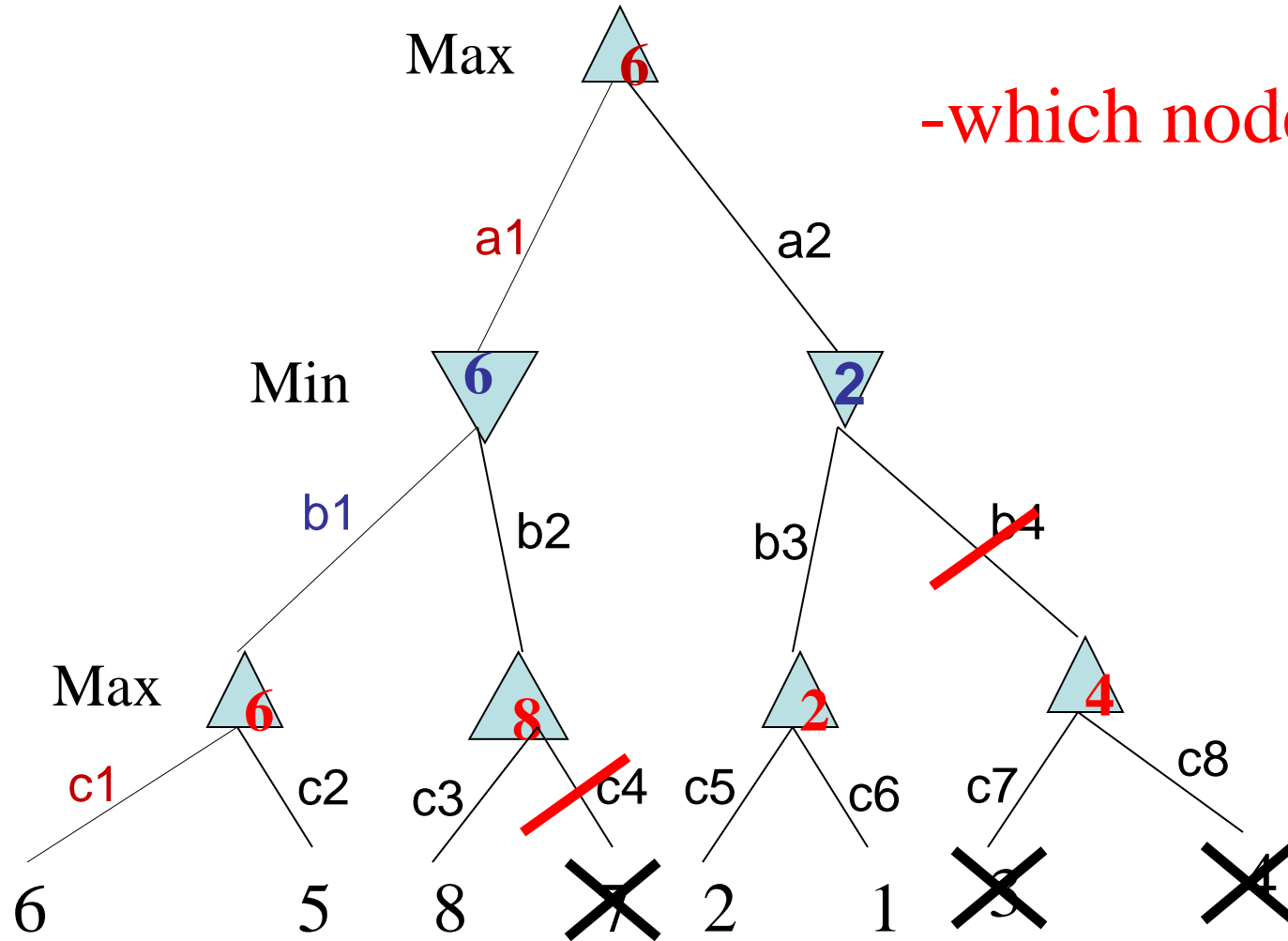


Example



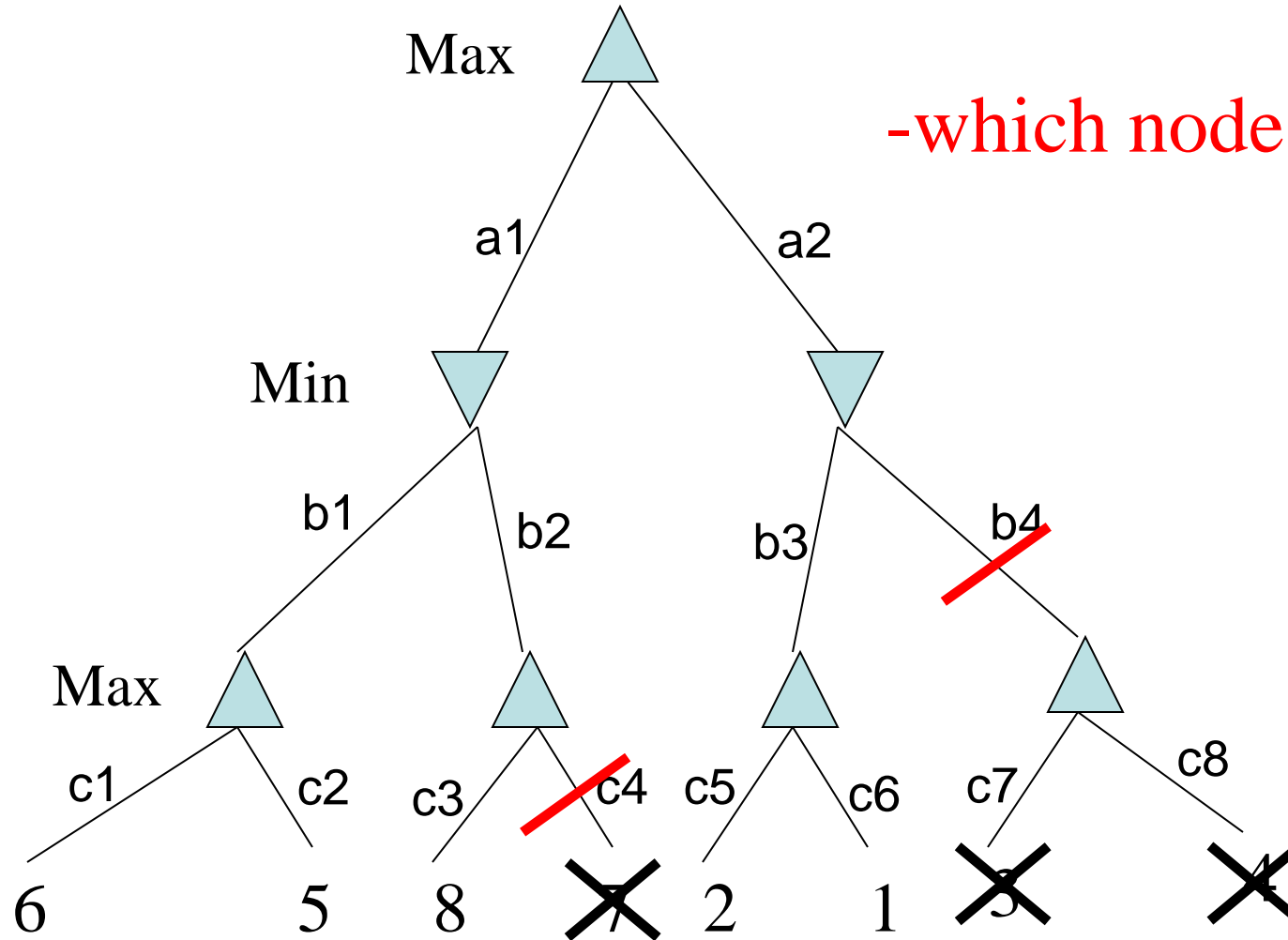
Example

-which nodes can be pruned?

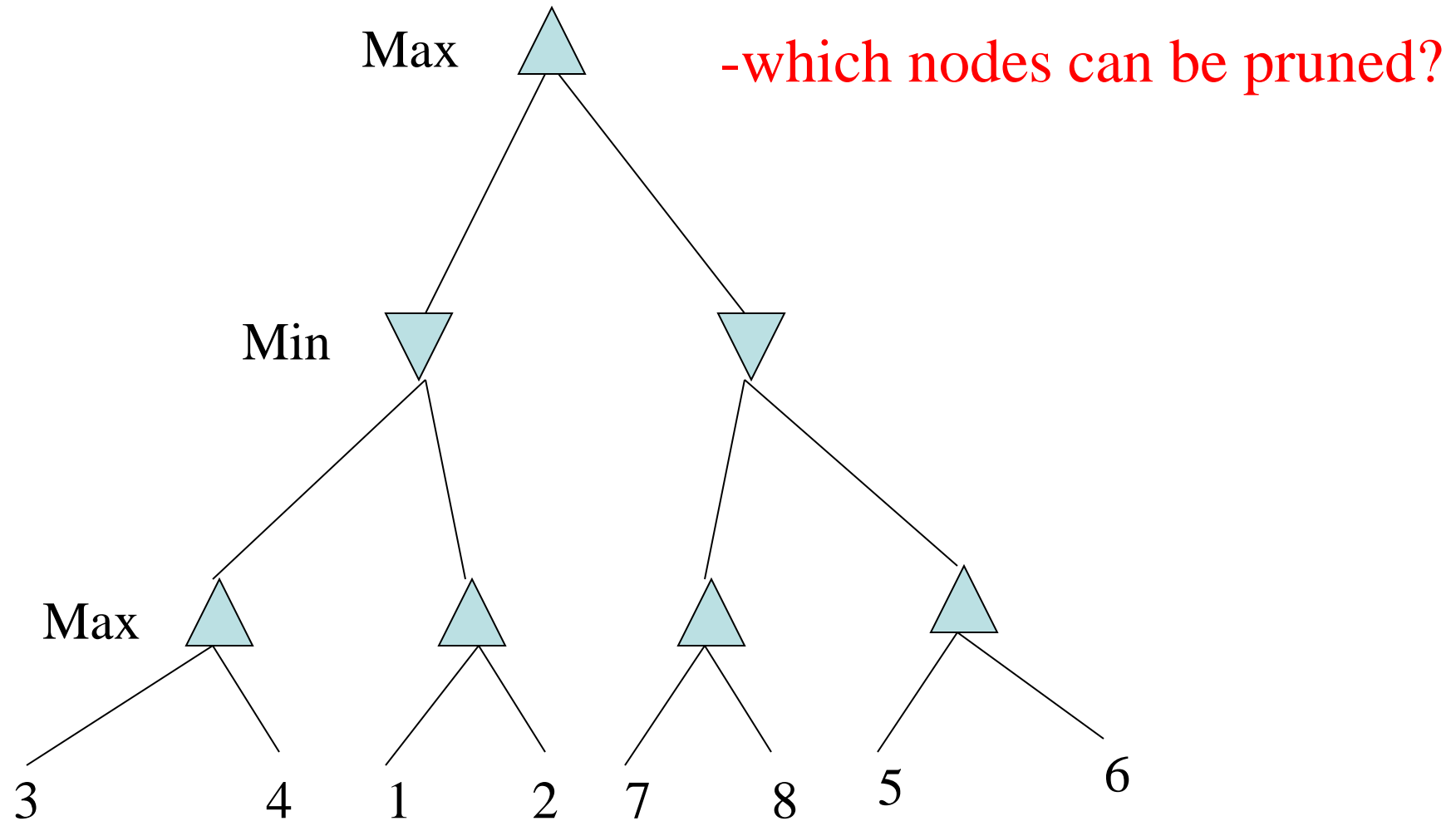


Example

-which nodes can be pruned?



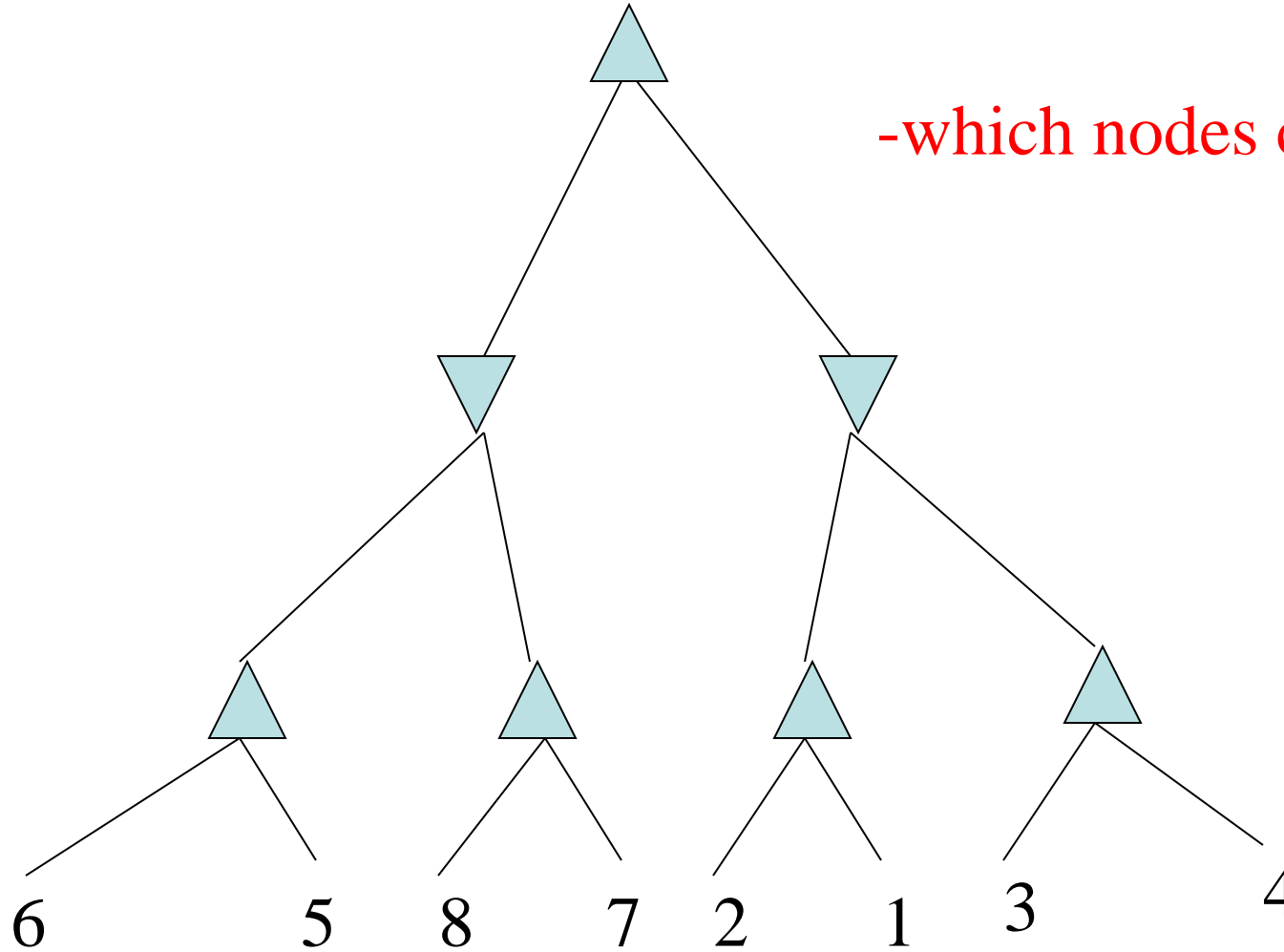
Answer to Example



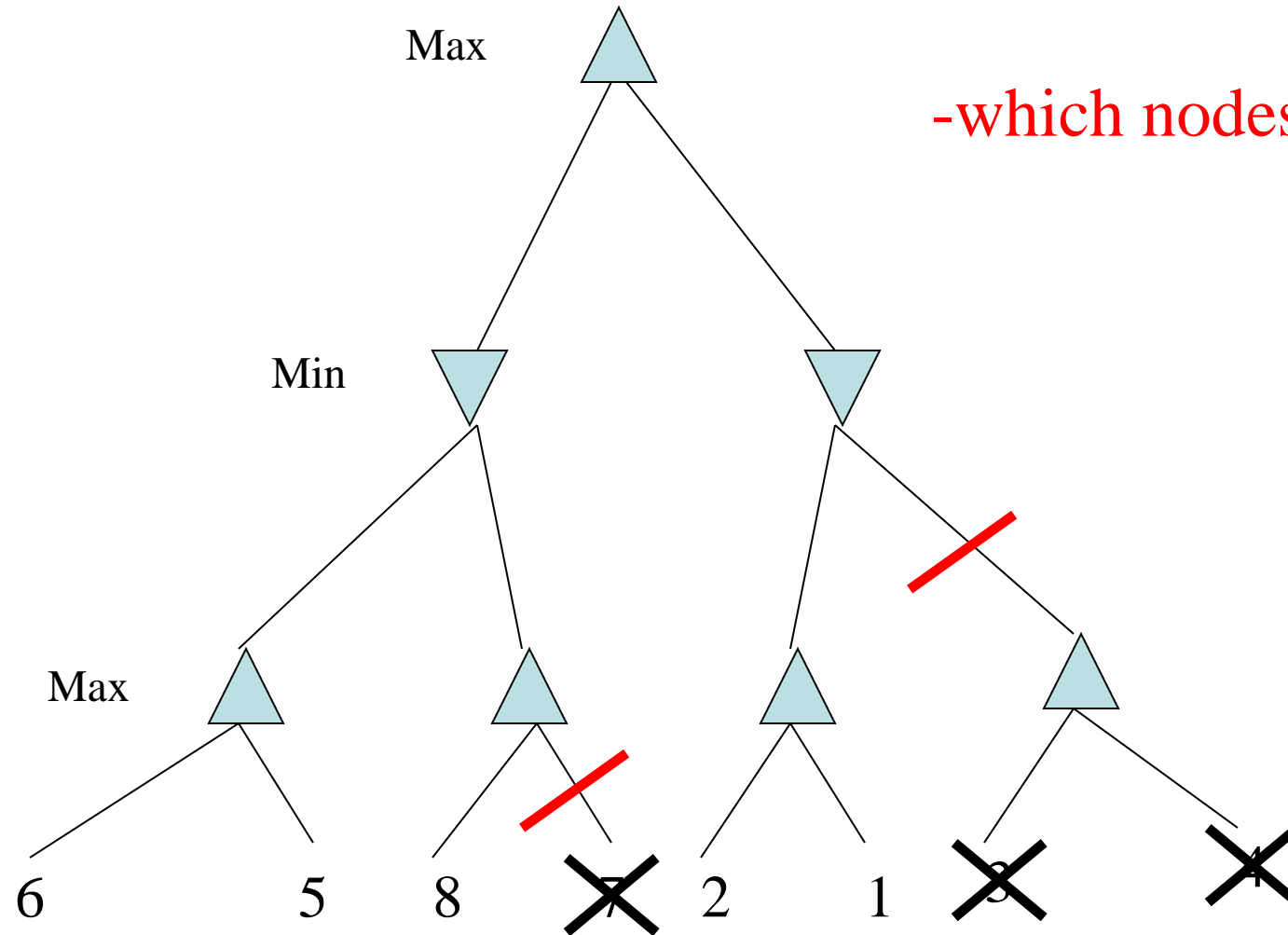
Answer: **NONE!** Because the most favorable nodes for both are explored **last** (i.e., in the diagram, are on the right-hand side).

Second Example

-which nodes can be pruned?



Answer to Second Example



Answer: **LOTS!** Because the most favorable nodes for both are explored **first** (i.e., in the diagram, are on the left-hand side).

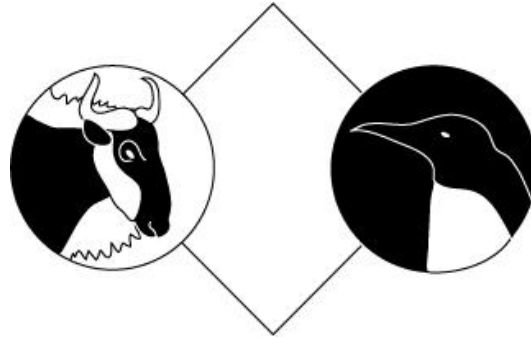
Deep Blue Algorithm

- AlphaBeta Pruning
- **Opening** Database of opening moves
- **Endgame** Database of all positions with five or fewer pieces



AlphaBeta in Go

- GNUGo
- In CGOS, GNUGo benchmark
- 业余5~10级左右
- <http://www.gnu.org/software/gnugo/>



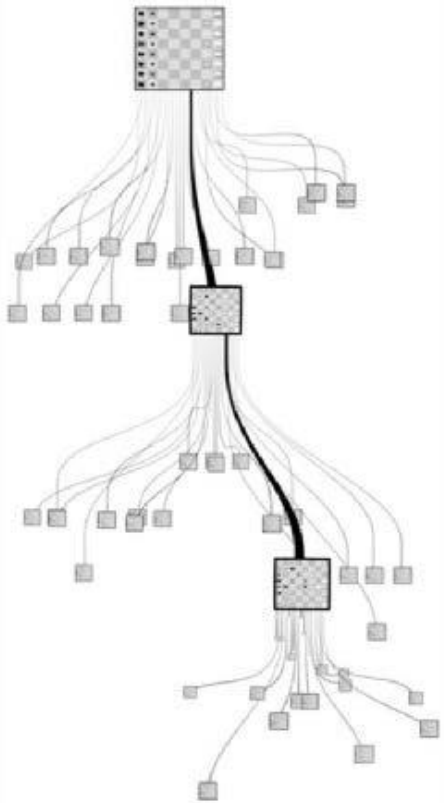
White (O) has captured 0 pieces
Black (X) has captured 0 pieces

	A	B	C	D	E	F	G	H	J	K	L	M	N	O	P	Q	R	S	T	
19	19
18	18
17	17
16	.	.	+	+	+	.	.	.	16
15	15
14	14
13	13
12	12
11	11
10	.	.	+	+	+	.	.	.	10
9	9
8	8
7	7
6	6
5	5
4	.	.	+	+	+	.	.	.	4
3	3
2	2
1	1
	A	B	C	D	E	F	G	H	J	K	L	M	N	O	P	Q	R	S	T	

black(1): ■

Go vs. Chess

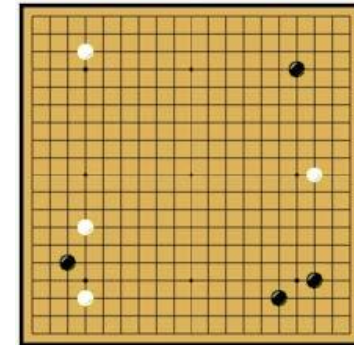
- Go has long been viewed as one of most complex game and most challenging of classic games for AI.



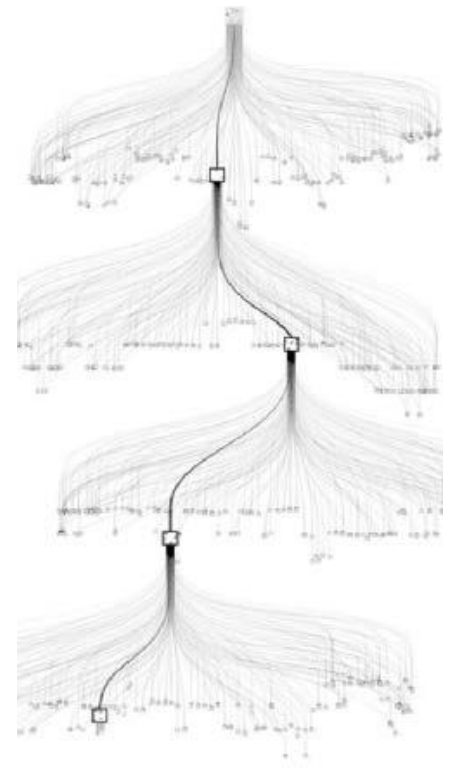
Chess ($b \approx 35$, $d \approx 80$)

$8 \times 8 = 64$, possible games $\approx 10^{120}$

Go ($b \approx 250$, $d \approx 150$)



$19 \times 19 = 361$, possible games $\approx 10^{170}$

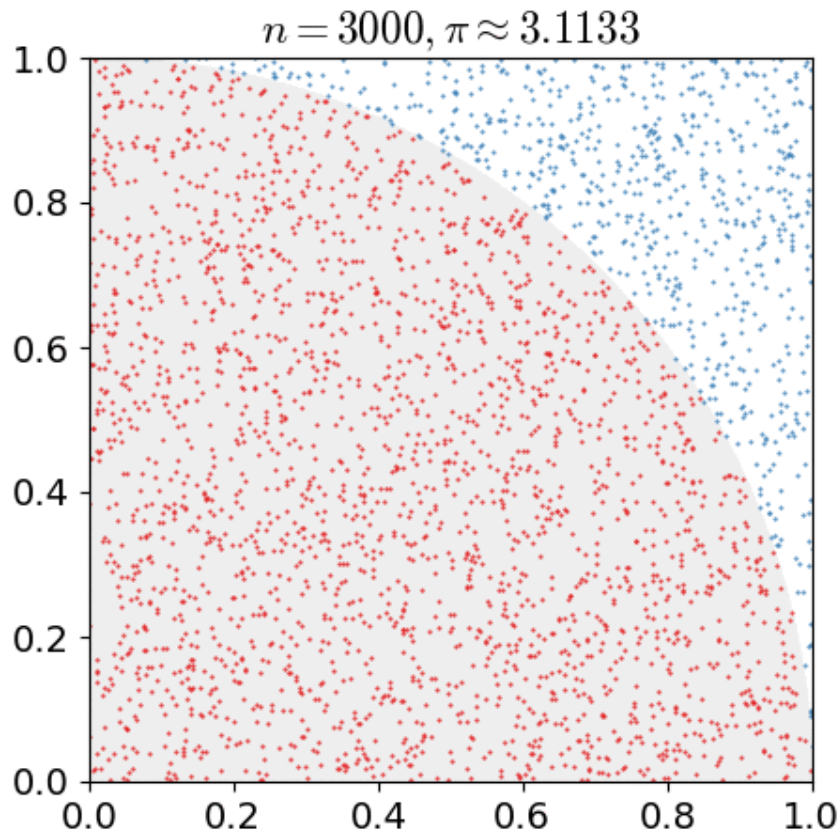




Monte Carlo Tree Search

MCTS

- Rely on **repeated random sampling** to obtain numerical results.
- *Example:* Approximating π by Monte Carlo Method



Given that circle and square have a ratio of areas that is $\pi/4$, the value of π can be approximated using a Monte-Carlo method:

- a) Draw a square on the ground, then inscribe a circle within it.
- b) Uniformly scatter some objects of uniform size over the square.
- c) Count the number of objects inside the circle and the square.
- d) The ratio of the two counts is an estimate of the ratio of the two areas, which is $\pi/4$. Multiply the result by 4 to estimate π .

```
1 import random
2
3 N = 50000
4 count = 0 # 将count当作Nx (即落入四分之一圆内的点)
5 for i in range(0, N):
6     x = random.uniform(0, 1)
7     y = random.uniform(0, 1)
8     if (x*x + y*y) < 1:
9         count = count+1
10
11 pi = 4*count/N
12 print("当模拟落点", N, "次时, pi的值为: ", pi)
13
```

Run: trypi x

D:\anaconda3\python.exe E:/browser_down/expert-system-master/expert-system-master/trypi.py
当模拟落点 50000 次时, pi的值为: 3.1464
进程已结束, 退出代码 0

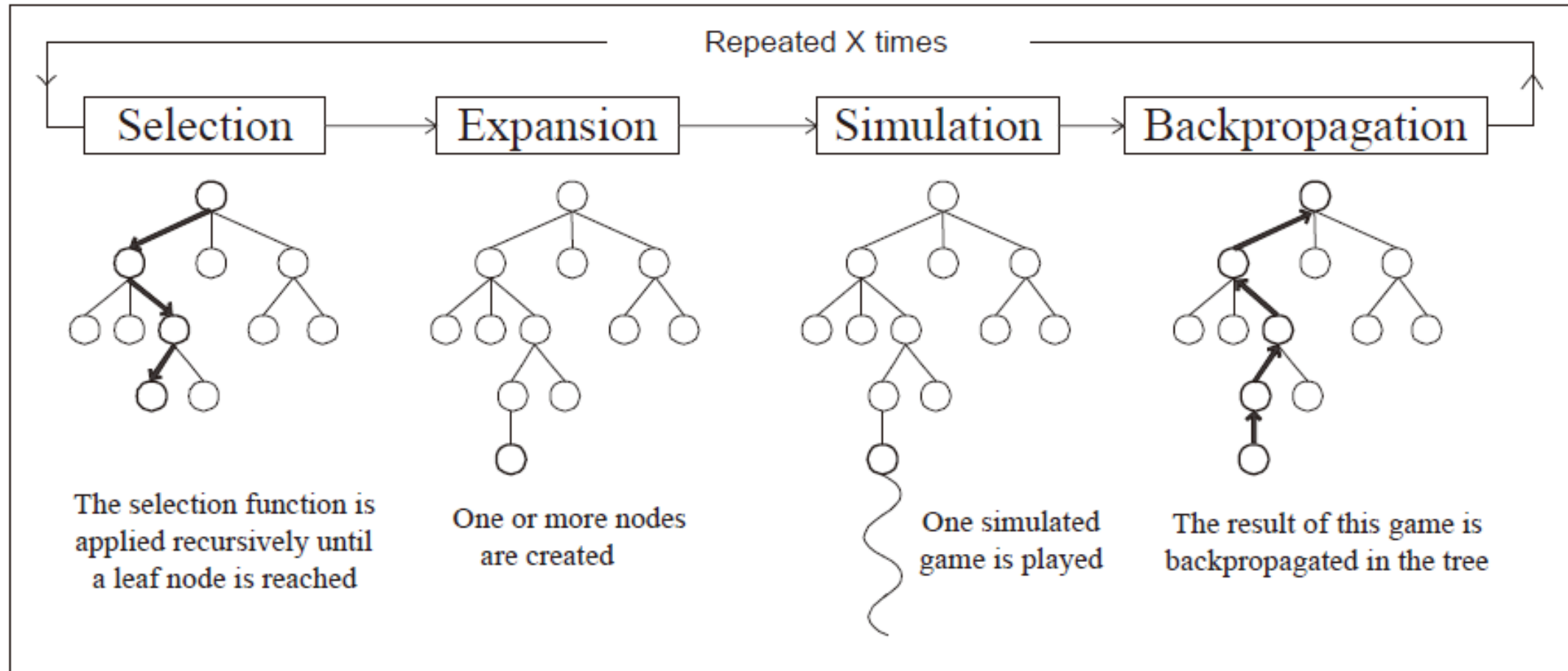
D:\anaconda3\python.exe E:/browser_down/expert-s
当模拟落点 100000 次时, pi的值为: 3.14608

D:\anaconda3\python.exe E:/browser_down/expert-sy
当模拟落点 1000000 次时, pi的值为: 3.14346

D:\anaconda3\python.exe E:/browser_down/expert-sys
当模拟落点 10000000 次时, pi的值为: 3.1418616

MCTS

- Four Steps to develop the game tree



MCTS 的具体步骤

1. Selection（选择）

- 从根节点开始，根据某种策略依次选择最佳的子节点，直到到达叶子节点。选择节点的好坏直接影响搜索的好坏，目前广泛采用的策略是uct算法（Upper Confidence Bound Apply to Tree）

2. Expansion（扩展）

- 扩展叶子节点，将一个或多个可行的落子添加为该叶子节点的子节点。

3. Simulation（模拟）

- 根据某种策略（比如围棋中的完全随机落子）从扩展的位置进行到游戏结束。模拟总是会产生一个结果，对于围棋类游戏来说就是获胜、失败或平局，但是广义上来说模拟的合法结果可以是任意值。

4. Backpropagation（反向传播）

- 将模拟的结果沿着传递路径反向传递回根节点。

UCT算法 (Upper Confidence Bound applied to Trees)

“Selection is the strategic task that selects one of the children of a given node. It controls the balance between **exploitation** and **exploration**.”

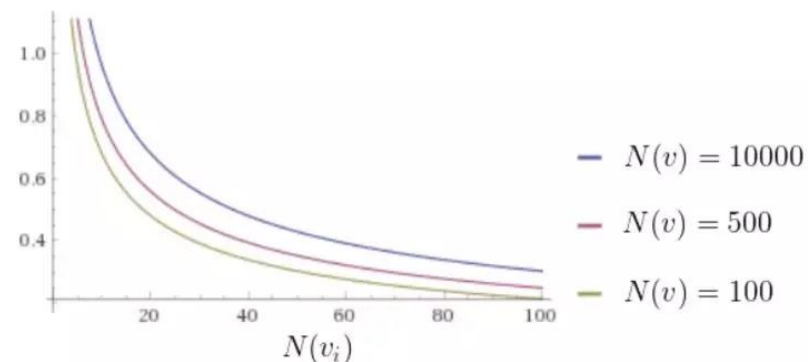
Exploitation(利用): 给定过去的经验选择能期望产生好的回报的动作。

Exploration(探索): 尝试可能能够使得在未来做出更好决策的新事物。

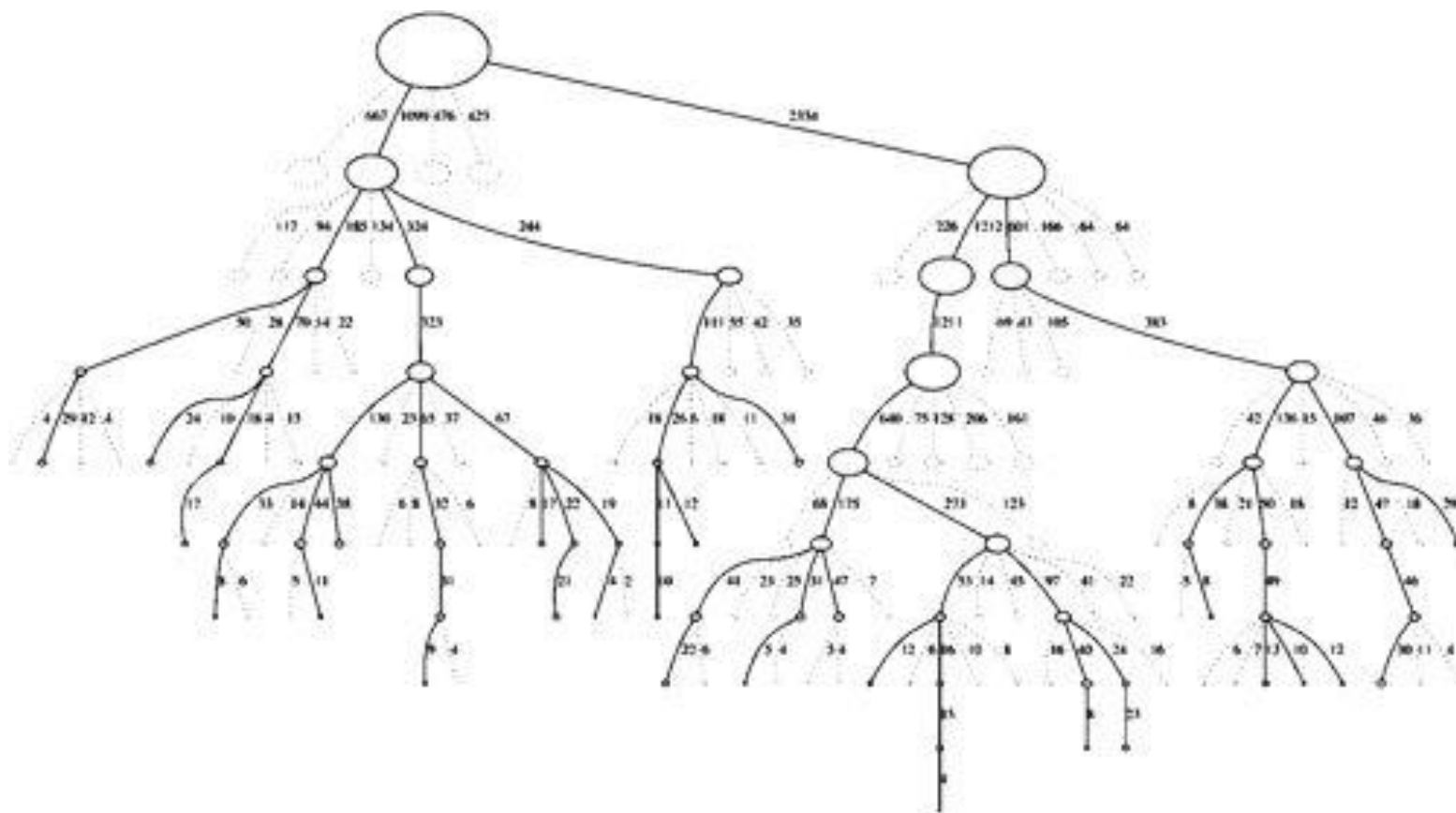
在选择节点的时候, 我们目前来讲当然应该考虑收益较高的节点; 但同时也要考虑那些由于被探测数量少, 暂时收益不高, 但在未来很有很有希望的节点.

$$\text{UCT}(v_i, v) = \frac{Q(v_i)}{N(v_i)} + c \sqrt{\frac{\ln(N(v))}{N(v_i)}}$$

公式的第一部分表示截至目前 v_i 节点平均每次的收益
公式的第二部分则倾向于那些相对较少被探索的节点

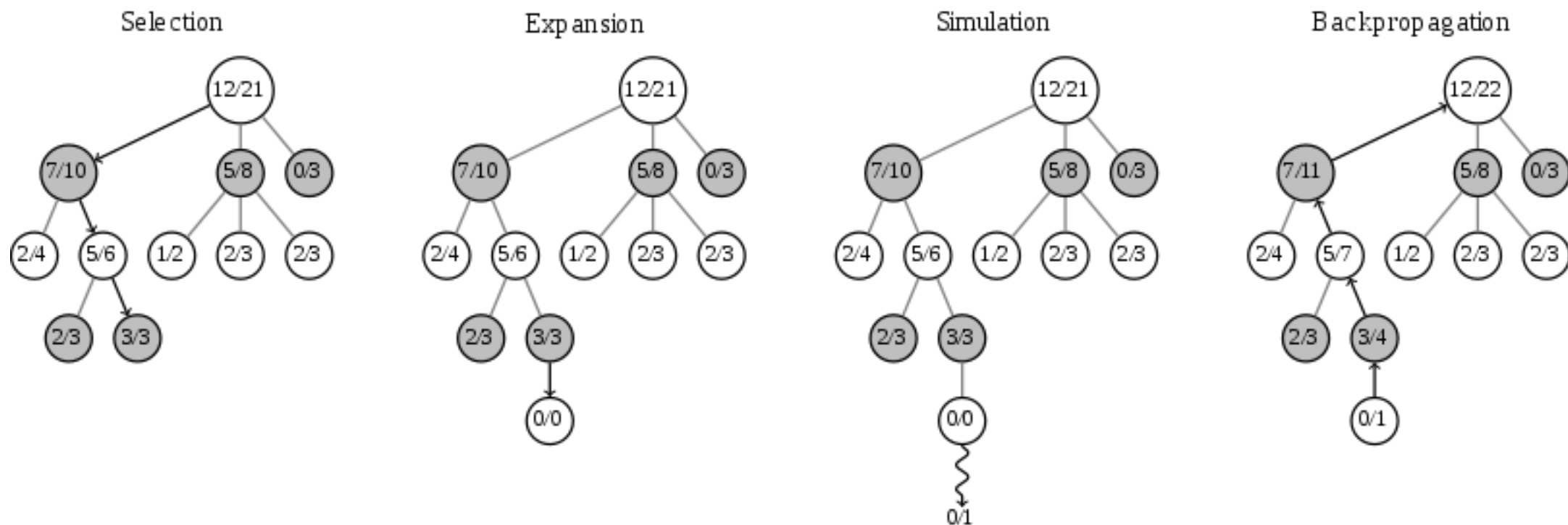


Asymmetric (非对称的建树过程)



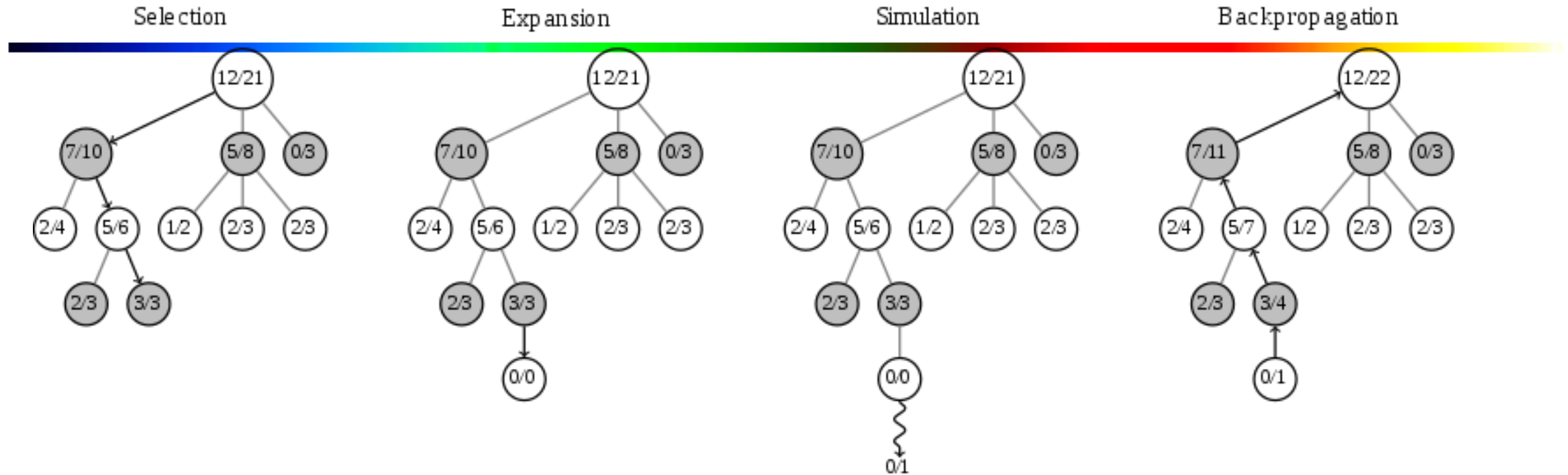
将更多算力用于探索未来发展更加优秀的分支上
也就是说MCTS可以找到那些更加优化的行动，并将搜索的工作聚焦在这些部分

wiki上的一个MCTS例子



每个节点（代表不同的局面）都有两个值，代表这个节点以及它的子节点模拟的次数和赢的次数，比如模拟了21次，赢了12次，记为 12/21。

这两个值也分别对应着最原始MCTS中的 $Q(v)$ 以及 $N(v)$ ——进行一局比赛 $N(v)+1$ ；赢一局 $Q(v)+1$ ，否则不变



- $\text{Score}(7/10) = \text{uct}(7/10, 12/21) = 7/10 + C \sqrt{\frac{\ln(21)}{10}} = 0.7 + 0.55C$
- $\text{Score}(5/8) = \text{uct}(5/8, 12/21) = 5/8 + C \sqrt{\frac{\ln(21)}{5}} = 0.625 + 0.62C$
- $\text{Score}(0/3) = \text{uct}(0/3, 12/21) = 0/3 + C \sqrt{\frac{\ln(21)}{3}} = 0 + 1.00C$

.....

c 越大越倾向于广度搜索，也就是探索有潜力的节点； c 越小越倾向于深度搜索，也就是多访问在当前已知信息下，平均奖励最高的节点。

MCTS in Go

- **MoGo**第一个使用使用蒙特卡洛树搜索的围棋程序(2006年), 在 9×9 的棋盘上击败了职业选手
- **DeepZenGo**是**AlphaGo**之前最强的围棋程序之一, 可以达到与职业棋士差距3~4子的水平



Algorithm of AlphaGo

- Deep neural networks
 - value networks: used to evaluate board positions
 - policy networks: used to select moves.
- **Monte-Carlo tree search (MCTS)**
 - Combines Monte-Carlo simulation with value networks and policy networks.
- Reinforcement learning
 - used to improve its play

*Source: Mastering Go with deep networks and tree search
Nature, Jan. 28, 2016*



Compared

软件或人类	BayesElo
AlphaGo Zero (40 blocks版)	5422?
AlphaGo (Master版)	5231?
AlphaGo Zero (20 blocks版)	5022?
AlphaGo (Lee版)	4672?
朴廷桓	4592?
柯洁	4590?
井山裕太	4546?
李世乭	4514?
DeepZenGo	4269
AlphaGo (Fan版, 176 GPU)	4122?
AlphaGo (Fan版, 48 CPU与8 GPU)	3862?
GNU Go	1800

AI Challenges

Alpha President vs Donalt Trump



**MAKE
COMPUTER
INTELLIGENT FOREVER!**

**MAKE
AMERICA
GREAT AGAIN!**

Summary

- Adversarial Search Methods
 - Minimax Search
 - Alpha-Beta Pruning
- Monte-Carlo Tree Search

Reference

