

CS 331: Artificial Intelligence


Adversarial Search

1

Games we will consider

- Deterministic
- Discrete states and decisions
- Finite number of states and decisions
- Perfect information i.e. fully observable
- Two agents whose actions alternate
- Their utility values at the end of the game are equal and opposite (we call this zero-sum)

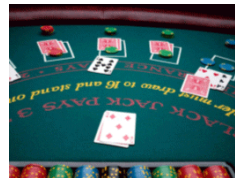
"It's not enough for me to win, I have to see my opponents lose"



2

Which of these games fit the description?

Two-player, zero-sum, discrete, finite, deterministic games of perfect information



What makes games hard?

- Hard to solve e.g. Chess has a search graph with about 10^{40} distinct nodes
- Need to make a decision even though you can't calculate the optimal decision
- Need to make a decision with time limits

Formal Definition of a Game

A quintuplet (S, I, Succ(), T, U):

| | |
|--------|--|
| S | Finite set of states. States include information on which player's turn it is to move. |
| I | Initial board position and which player is first to move |
| Succ() | Takes a current state and returns a list of (move,state) pairs, each indicating a legal move and the resulting state |
| T | Terminal test which determines when the game ends. Terminal states: subset of S in where the game has ended |
| U | Utility function (aka objective function or payoff function): maps from terminal state to real number |

5

Nim

Many different variations. We'll do this one.

- Start with 9 beaver logos
- In one player's turn, that player can remove 1, 2 or 3 beaver logos
- The person who takes the last beaver logo wins

6

Nim



7

Formal Definition of Nim

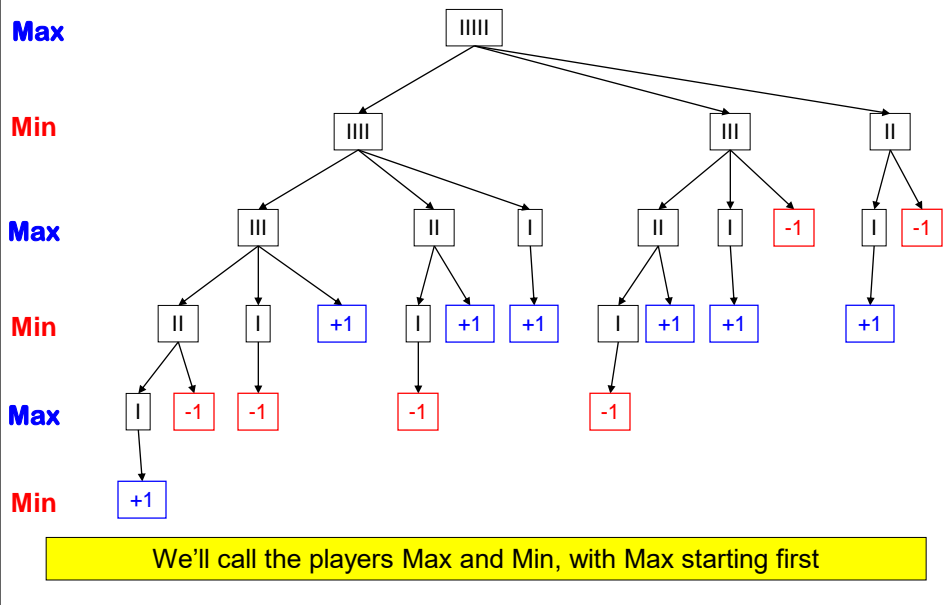
A quintuplet (S, I, Succ(), T, U):

Notation: Max(IIII)
 Who's move # matches left

| | |
|--------|---|
| S | Max(IIII), Max(III), Max(II), Max(I) Min(IIII), Min(III), Min(II), Min(I) |
| I | Max(IIII) |
| Succ() | Succ(Max(IIII)) = {Min(IIII), Min(III), Min(II)} Succ(Min(IIII)) = {Max(III), Max(II), Max(I)} Succ(Max(III)) = {Min(II), Min(I)} Succ(Min(III)) = {Max(II), Max(I)} Succ(Max(II)) = {Min(I)} Succ(Min(II)) = {Max(I)} |
| T | Max(I), Max(II), Max(III), Min(I), Min(II), Min(III) |
| U | Utility(Max(I) or Max(II) or Max(III)) = +1, Utility(Min(I) or Min(II) or Min(III)) = -1 |

8

Nim Game Tree



How to Use a Game Tree

- Max wants to maximize his utility
- Min wants to minimize Max's utility
- Max's strategy must take into account what Min does since they alternate moves
- A move by Max or Min is called a ply

The Minimax Value of a Node

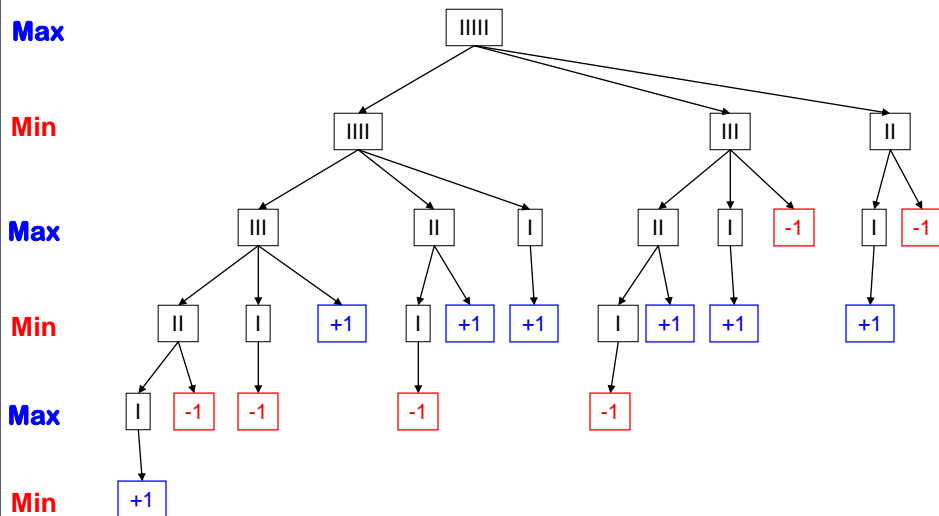
The minimax value of a node is the utility for MAX of being in the corresponding state, *assuming that both players play optimally* from there to the end of the game

MINIMAX - VALUE(n) =

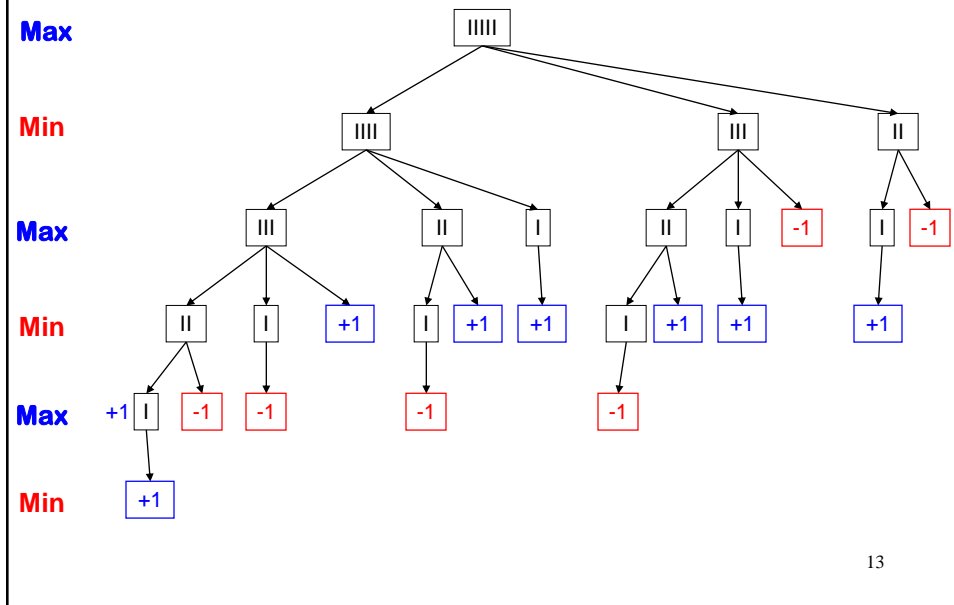
$$\left\{ \begin{array}{ll} \text{UTILITY}(n) & \text{If } n \text{ is a terminal state} \\ \max_{s \in \text{Successor}(n)} \text{MINIMAX - VALUE}(s) & \text{If } n \text{ is a MAX node} \\ \min_{s \in \text{Successor}(n)} \text{MINIMAX - VALUE}(s) & \text{If } n \text{ is a MIN node} \end{array} \right.$$

Minimax value maximizes worst-case outcome for MAX

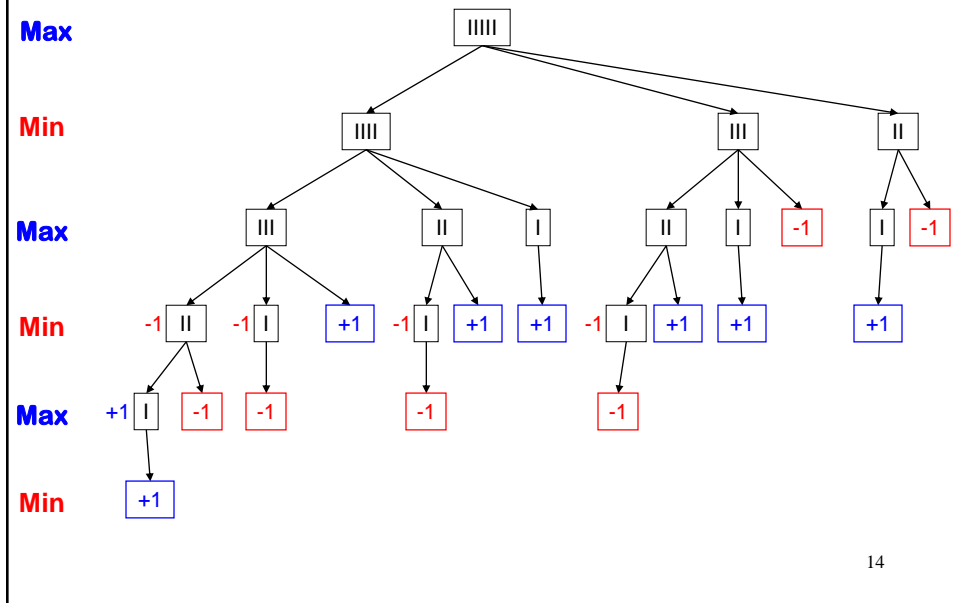
Nim Game Tree



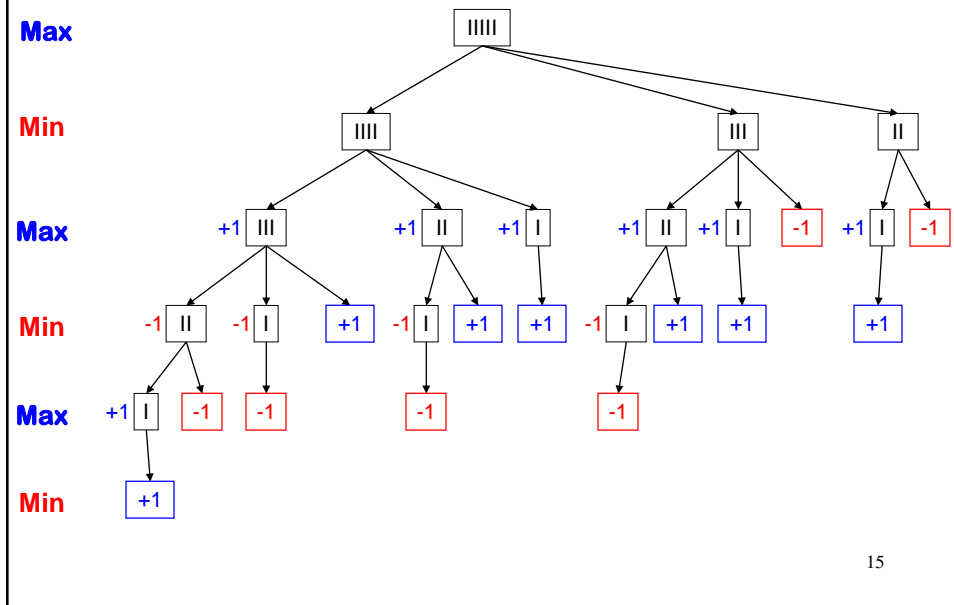
Minimax Values in Nim Game Tree



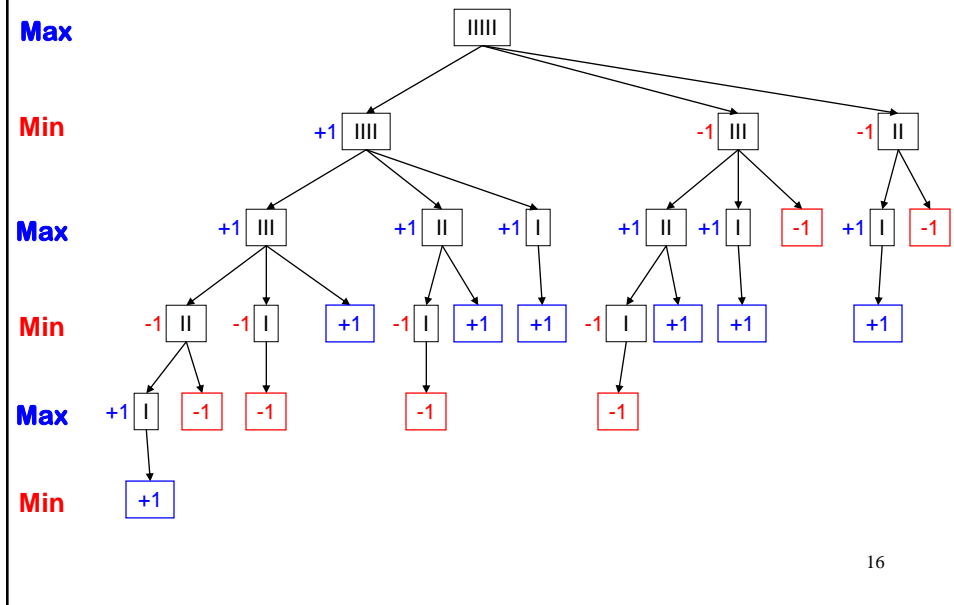
Minimax Values in Nim Game Tree



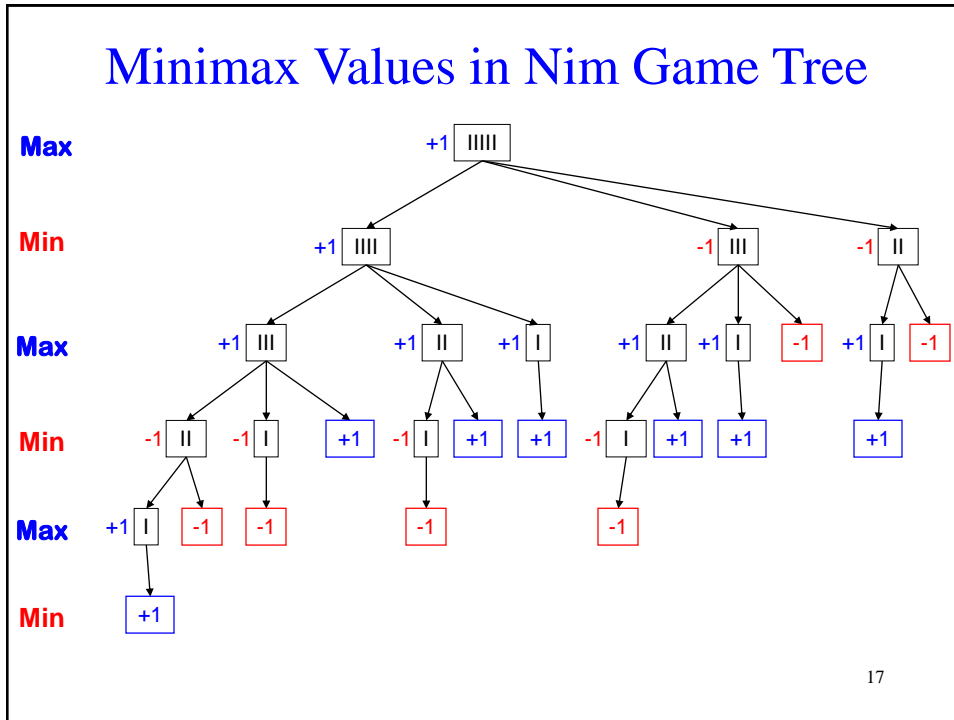
Minimax Values in Nim Game Tree



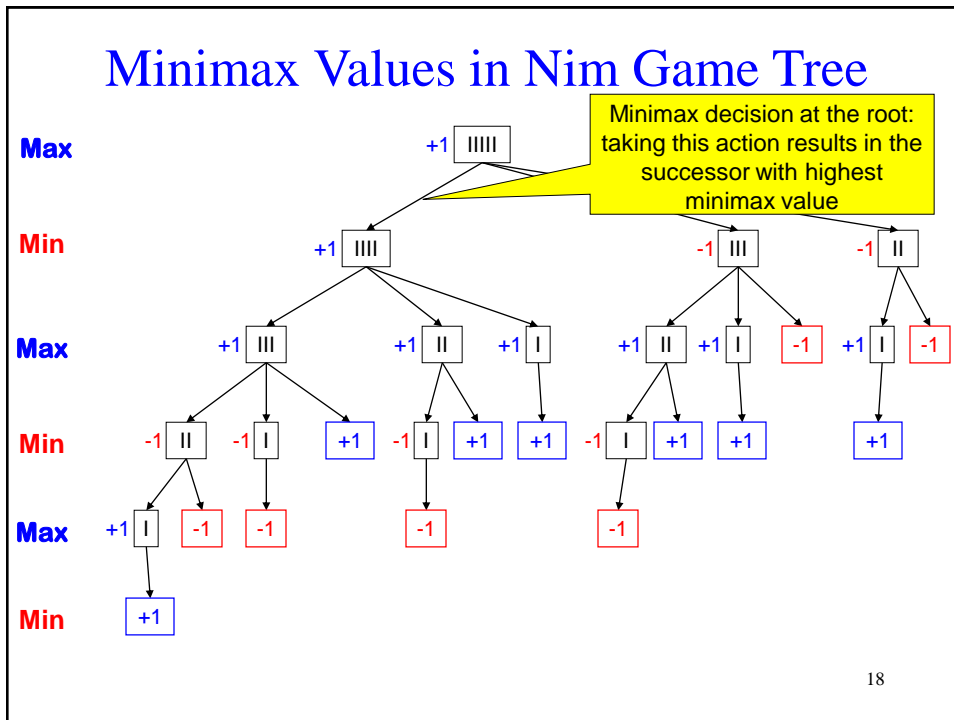
Minimax Values in Nim Game Tree



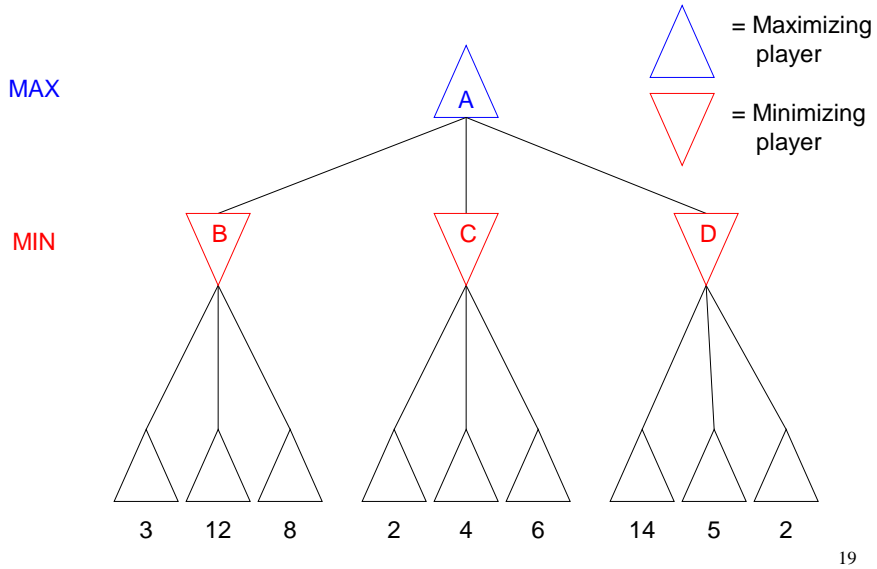
Minimax Values in Nim Game Tree



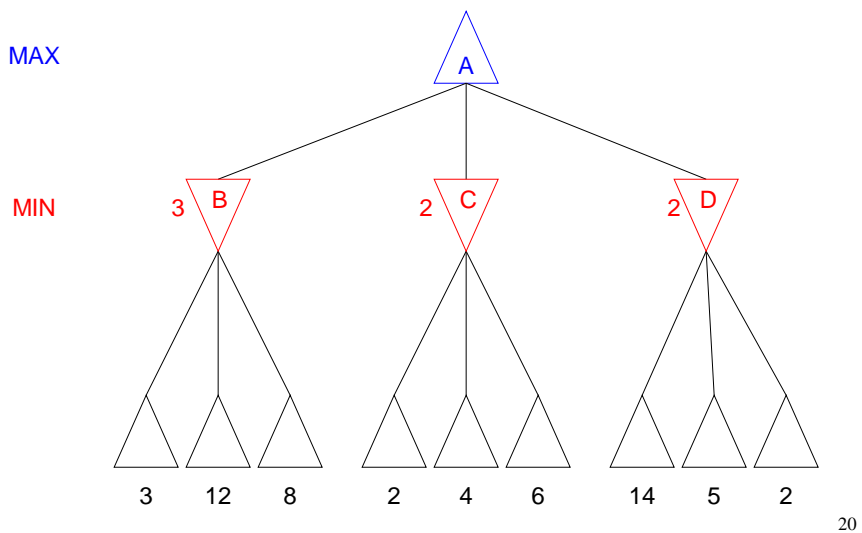
Minimax Values in Nim Game Tree



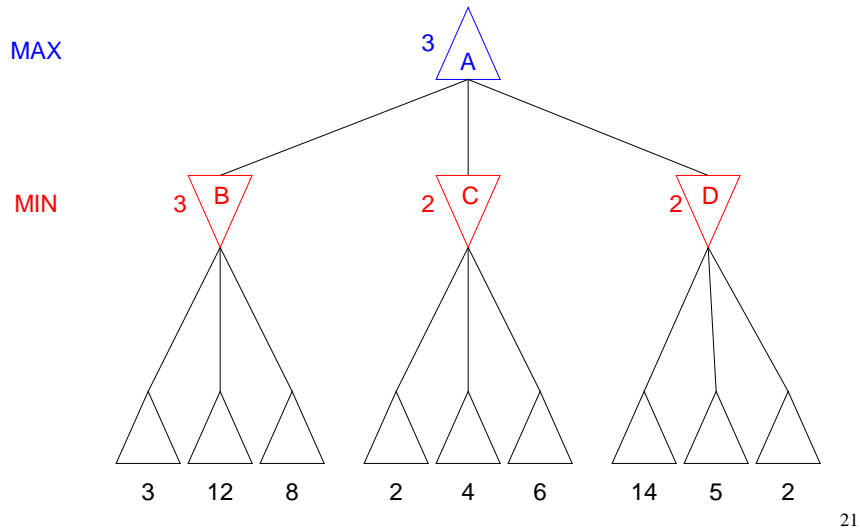
Another Example



Another Example



Another Example



The MINIMAX Algorithm

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*)

$v \leftarrow$ -Infinity

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*)

$v \leftarrow$ Infinity

for a, s in SUCCESSORS(*state*) **do**

$v \leftarrow$ MIN(v , MAX-VALUE(s))

return v

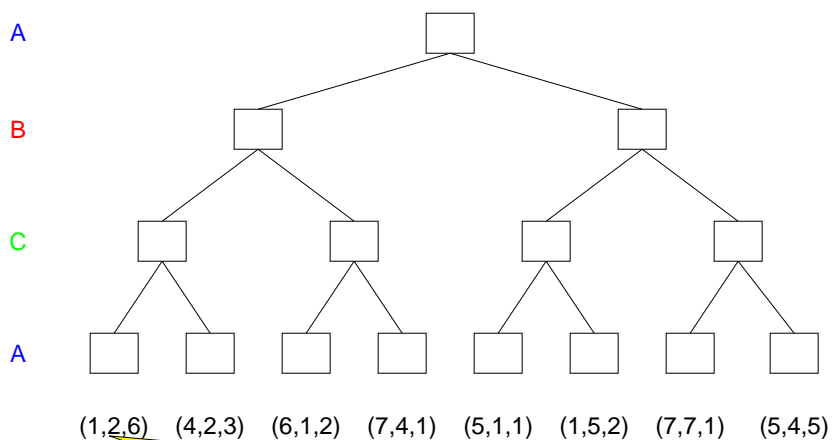
22

The MINIMAX algorithm

- Computes minimax decision from the current state
- Depth-first exploration of the game tree
- Time Complexity $O(b^m)$ where b =# of legal moves, m =maximum depth of tree
- Space Complexity:
 - $O(bm)$ if all successors generated at once
 - $O(m)$ if only one successor generated at a time (each partially expanded node remembers which successor to generate next)

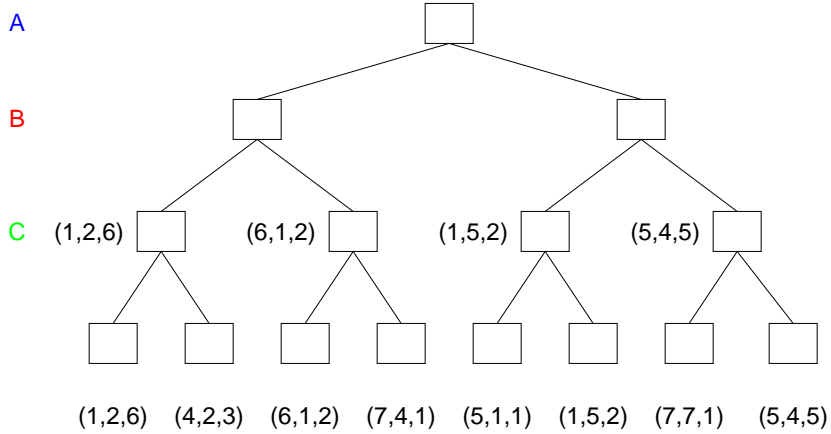
23

Minimax With 3 Players



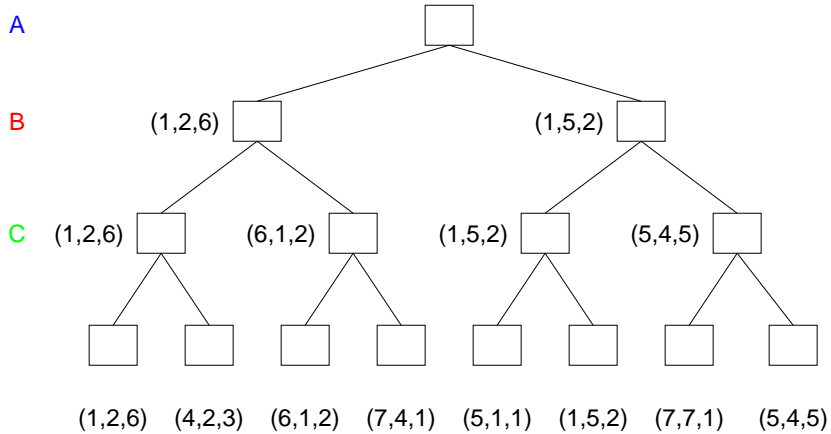
Now have a vector of utilities for players (A,B,C). All players maximize their utilities. Note: In two-player, zero-sum games, we have a single value because the values are always opposite.

Minimax With 3 Players



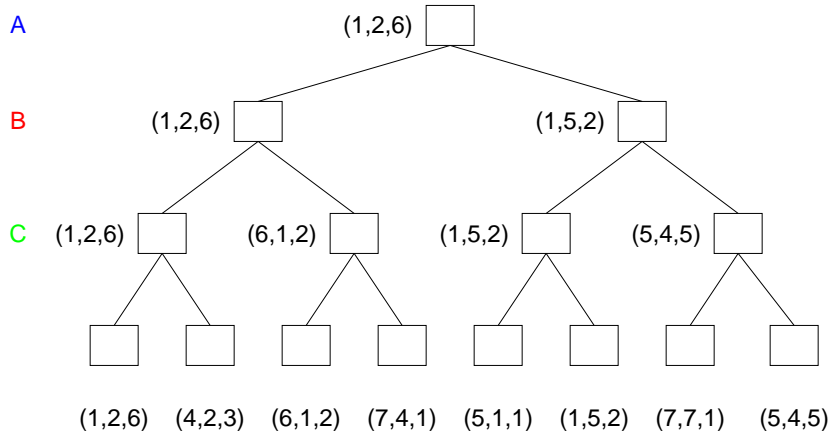
25

Minimax With 3 Players



26

Minimax With 3 Players



27

Subtleties With Multiplayer Games

- Alliances can be made and broken
- For example, if A and B are weaker than C, they can gang up on C
- But A and B can turn on each other once C is weakened
- But society considers the player that breaks the alliance to be dishonorable

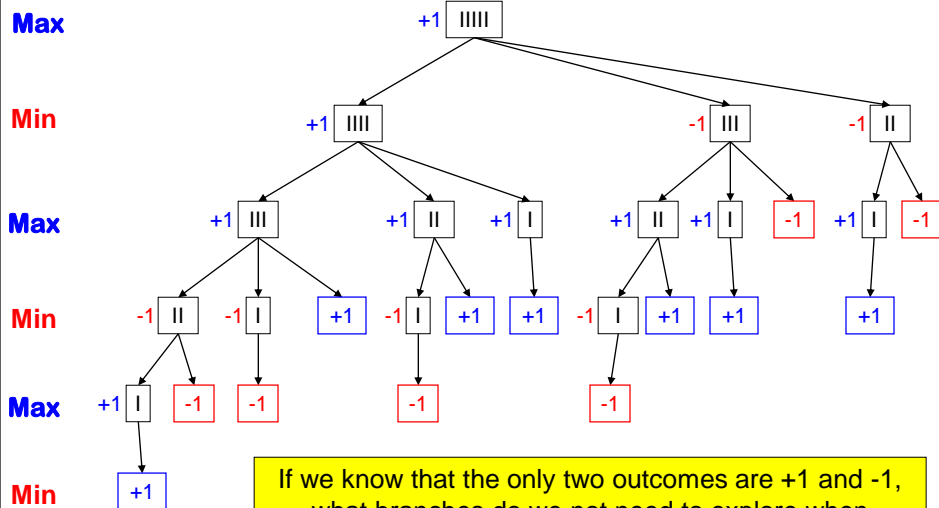
28

Pruning

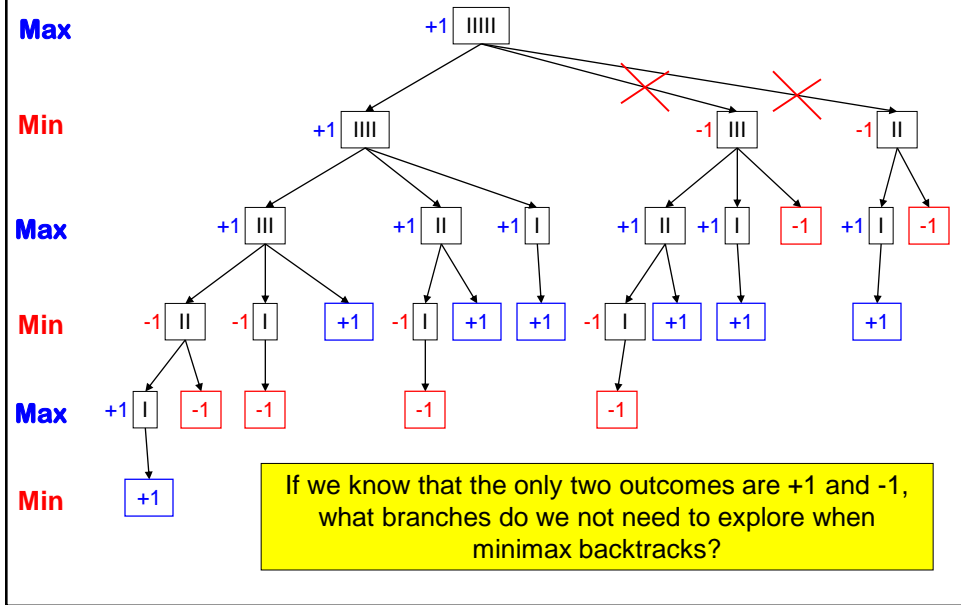
- Can we improve on the time complexity of $O(b^m)$?
- Yes if we prune away branches that cannot possibly influence the final decision

29

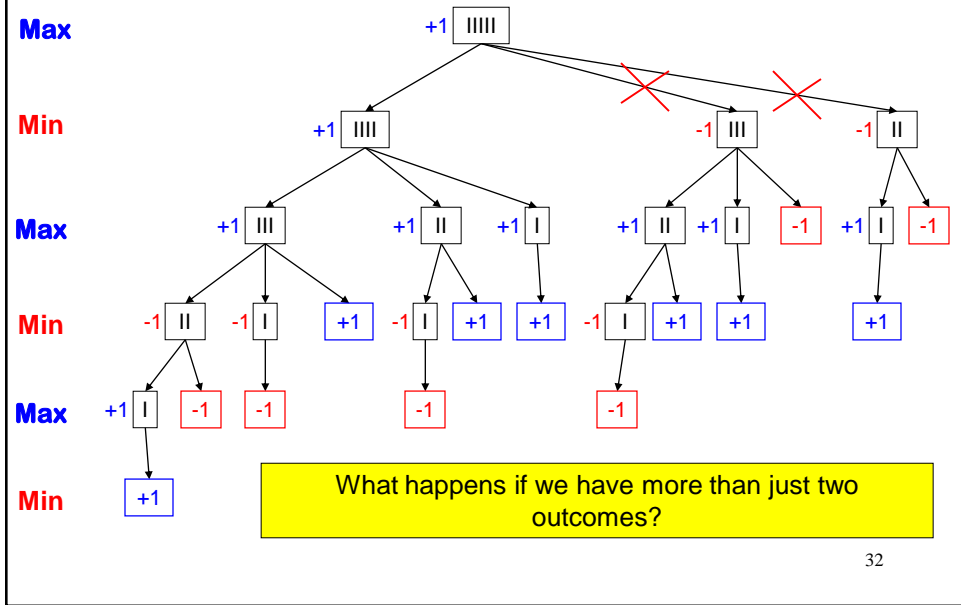
Pruning in Nim



Pruning in Nim



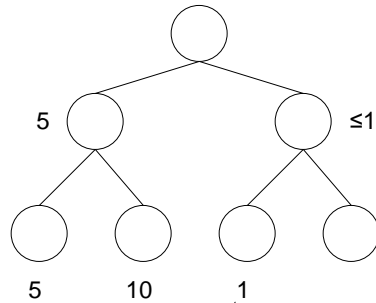
Pruning in Nim



Pruning Intuition (General Case)

MAX

MIN



The max player will never choose the right subtree once it knows that it is upper bounded by 1

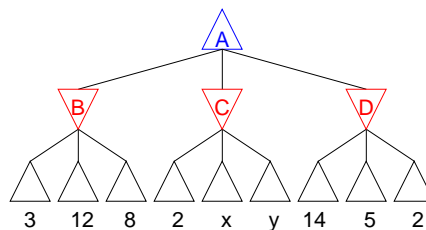
Suppose we just went down this branch. We know that the minimax value of its parent will be ≤ 1

33

Pruning Example

MAX

MIN



MINIMAX-VALUE(root)

$= \max(\min(3, 12, 8), \min(2, x, y), \min(14, 5, 2))$

$= \max(3, \min(2, x, y), 2)$

$= \max(3, z, 2)$ where $z \leq 2$

$= 3$

34

Pruning Intuition

Remember that minimax search is DFS.

At any one time, we only have to consider the nodes along a single path in the tree

In general, let:

- α = highest minimax value of all of the MAX player's choices expanded on current path (best score for MAX so far)
- β = lowest minimax value of all of the MIN player's choices expanded on current path (best score for MIN so far)
- If at a MIN player node, prune if minimax value of node $\leq \alpha$
- If at a MAX player node, prune if minimax value of node $\geq \beta$

35

ALPHA-BETA Pseudocode

function ALPHA-BETA-SEARCH(*state*) **returns** an action

inputs: *state*, current state in game

$v \leftarrow \text{MAX-VALUE}(\text{state}, -\infty, +\infty)$

return the *action* in SUCCESSORS(*state*) with value *v*

function MAX-VALUE(*state*, α , β) **returns** a utility value

inputs: *state*, current state in game

α , the value of the best alternative for MAX along the path to *state*

β , the value of the best alternative for MIN along the path to *state*

if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)

$v \leftarrow -\infty$

for *a, s* in SUCCESSORS(*state*) **do**

$v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(s, \alpha, \beta))$

if $v \geq \beta$ **then return** *v*

$\alpha \leftarrow \text{MAX}(\alpha, v)$

return *v*

36

ALPHA-BETA Pseudocode

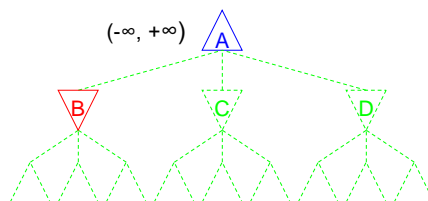
```
function MIN-VALUE(state,  $\alpha$ ,  $\beta$ ) returns a utility value
  inputs: state, current state in game
            $\alpha$ , the value of the best alternative for MAX along the path to state
            $\beta$ , the value of the best alternative for MIN along the path to state
  if TERMINAL-TEST(state) then return UTILITY(state)
   $v \leftarrow +\infty$ 
  for a, s in SUCCESSORS(state) do
     $v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(s, \alpha, \beta))$ 
    if  $v \leq \alpha$  then return  $v$ 
     $\beta \leftarrow \text{MIN}(\beta, v)$ 
  return  $v$ 
```

37

Illustrating the Pseudocode

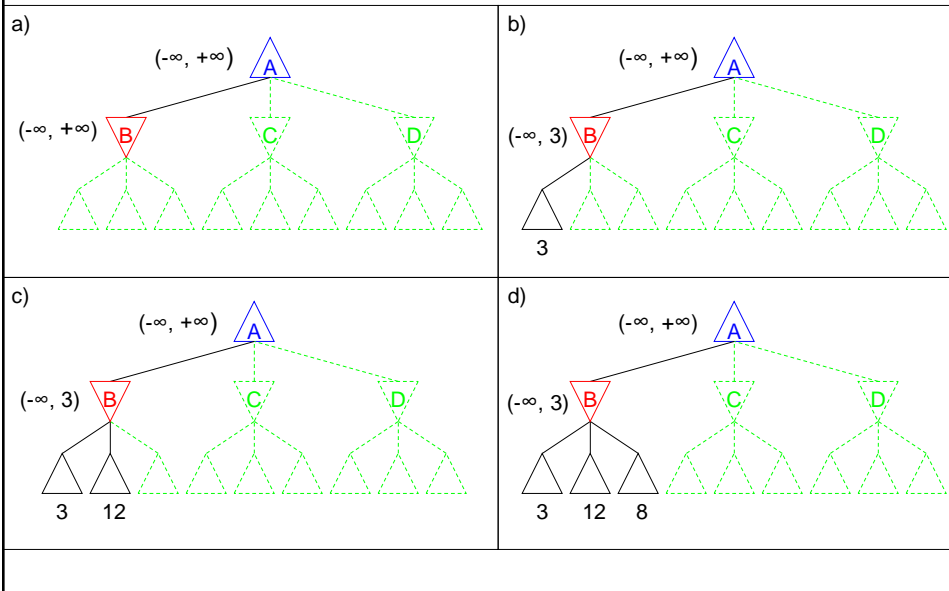
- In the example to follow, the notation $(-\infty, +\infty)$ represents the (α, β) values for the corresponding node
- This example is intended to illustrate how the actual **implementation** of Alpha-Beta pruning works

 = Maximizing player
 = Minimizing player

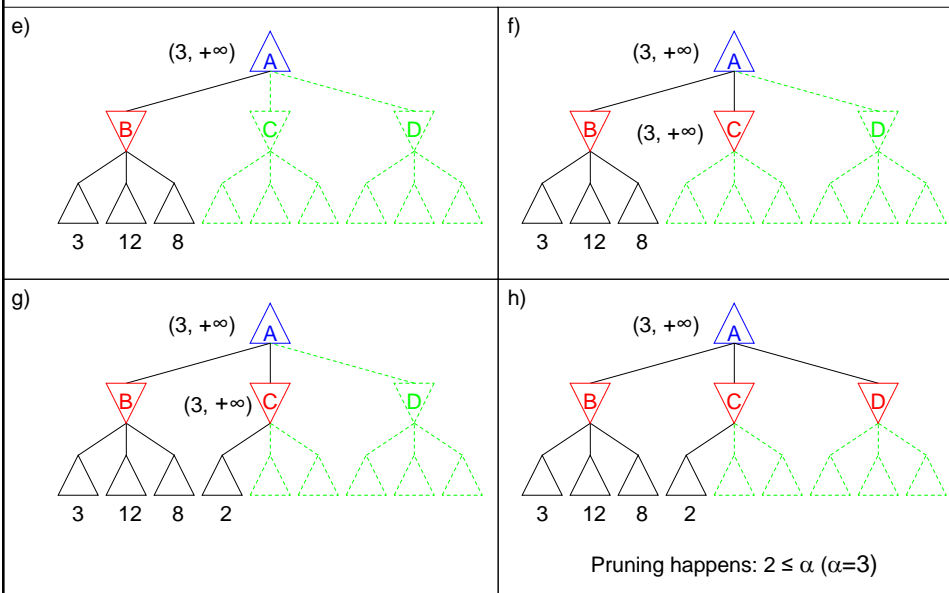


38

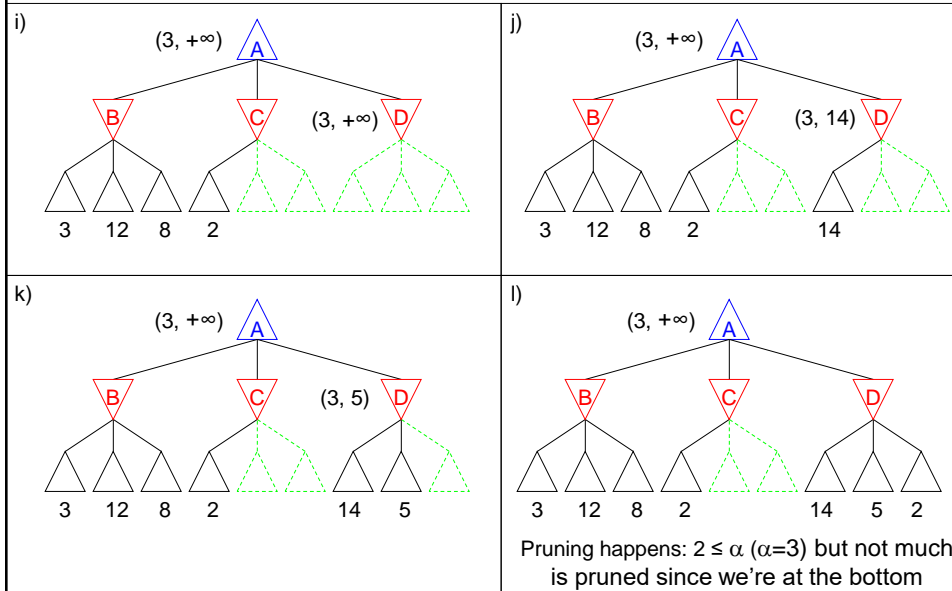
Alpha-Beta Pruning Example



Alpha-Beta Pruning Example



Alpha-Beta Pruning Example



Effectiveness of Alpha-Beta

- Depends on order of successors
- Best case: Alpha-Beta reduces complexity from $O(b^m)$ for minimax to $O(b^{m/2})$
- This means Alpha-Beta can lookahead about twice as far as minimax in the same amount of time

Implementation Details

- In games we have the problem of **transposition**
- Transposition means different permutations of the move sequence that end up in the same position
- Results in lots of repeated states
- Use a transposition table to remember the states you've seen (similar to closed list)

43

What you should know

- Be able to draw up a game tree
- Know how the Minimax algorithm works
- Know how the Alpha-Beta algorithm works
- Be able to do both algorithms by hand

44