CS 331: Artificial Intelligence
Adversarial Search

Games we will consider

- Deterministic
- Discrete states and decisions
- Finite number of states and decisions
- Perfect information ie. 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"

Which of these games fit the description?

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





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

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

Nim

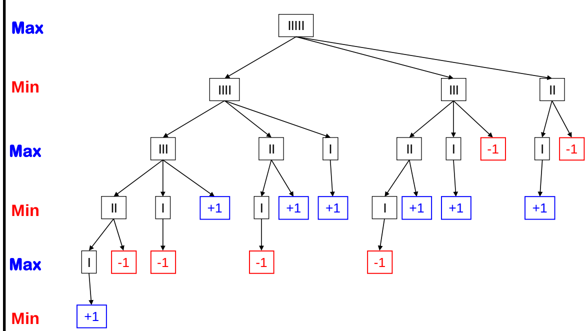
Formal Definition of Nim

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

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

S	Max(IIIII), Max(III), Max(II), Max(I) Min(IIII), Min(III), Min(II), Min(I)
I	Max(IIIII)
Succ()	Succ(Max(IIIII)) = {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

Nim Game Tree



We'll call the players Max and Min, with Max starting first

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

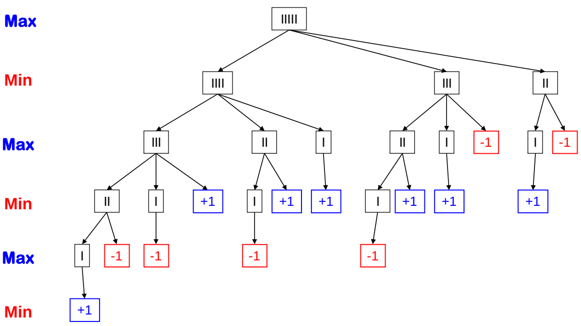
The Minimax Value of a Node

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

$$\text{MINIMAX-VALUE}(n) = \begin{cases} \text{UTILITY}(n) & \text{If n is a terminal state} \\ \max_{s \in Successors(n)} \text{MINIMAX-VALUE}(s) & \text{If n is a MAX node} \\ \min_{s \in Successors(n)} \text{MINIMAX-VALUE}(s) & \text{If n is a MIN node} \end{cases}$$

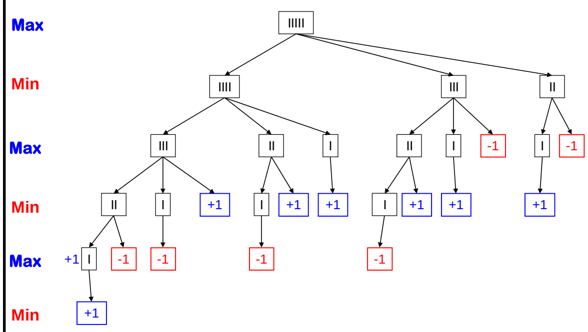
Minimax value maximizes worst-case outcome for MAX

Nim Game Tree



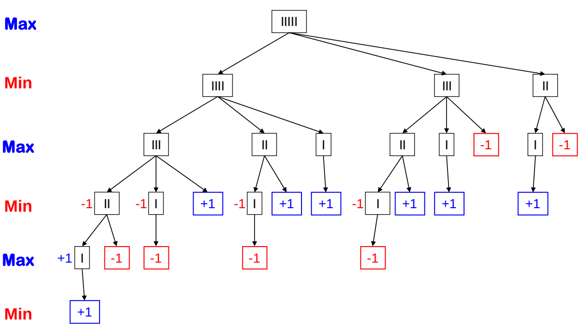
13

Minimax Values in Nim Game Tree



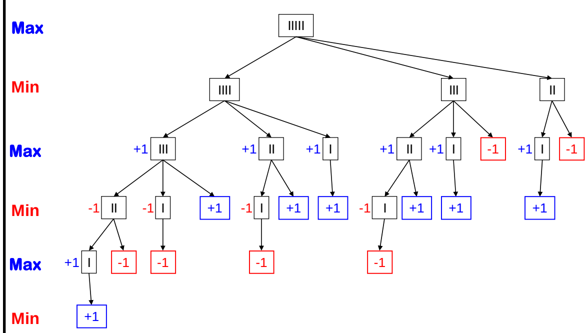
14

Minimax Values in Nim Game Tree



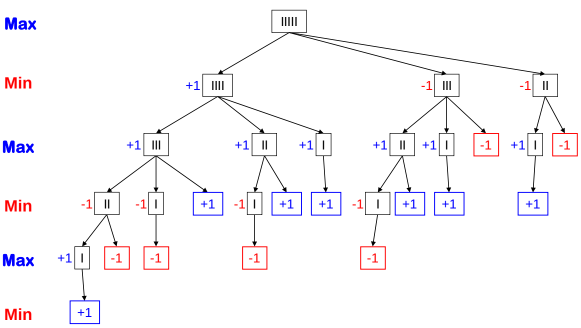
15

Minimax Values in Nim Game Tree



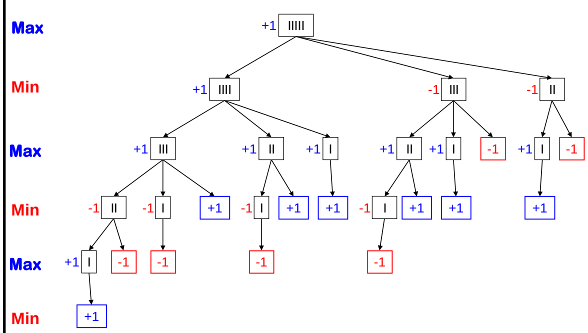
16

Minimax Values in Nim Game Tree



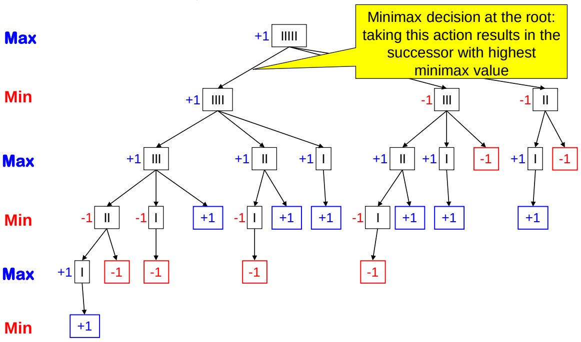
17

Minimax Values in Nim Game Tree

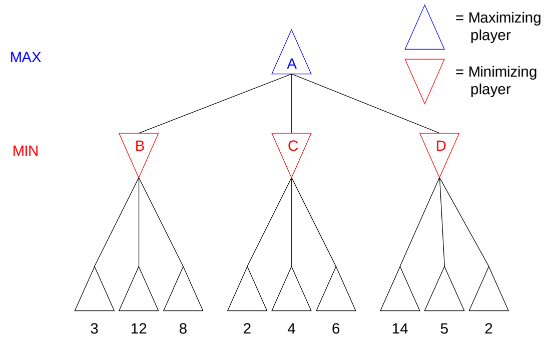


18

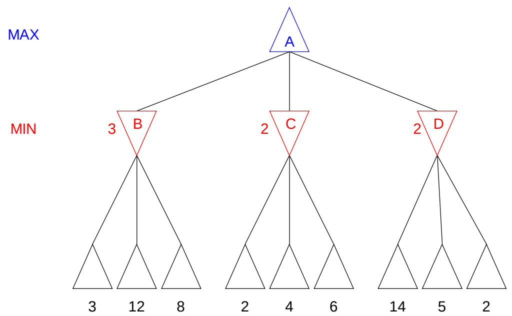
Minimax Values in Nim Game Tree



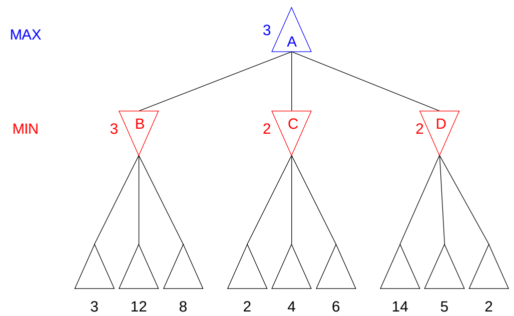
Another Example



Another Example



Another Example



The MINIMAX Algorithm

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

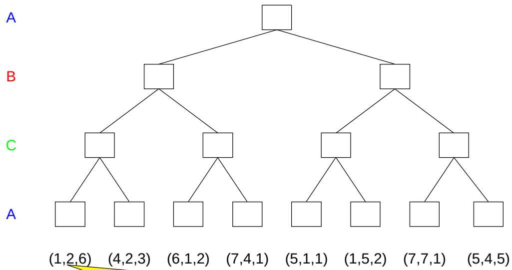
function MAX-VALUE(*state*) **returns** *a utility value*
  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  v ← - Infinity
  for *a*, *s* in SUCCESSORS(*state*) **do**
    v ← MAX(*v*, MIN-VALUE(*s*))
  return *v*

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

The MINIMAX algorithm

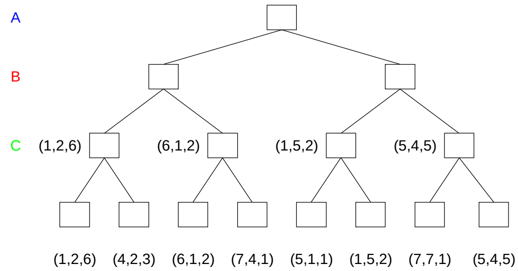
- Computes minimax decision from the current state
- Depth-first exploration of the game tree
- Time Complexity $O(b^m)$ where b=# of legal moves, m=maximum depth of tree
- Space Complexity:
  - O(bm) if all successors generated at once
  - O(m) if only one successor generated at a time (each partially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

Minimax With 3 Players



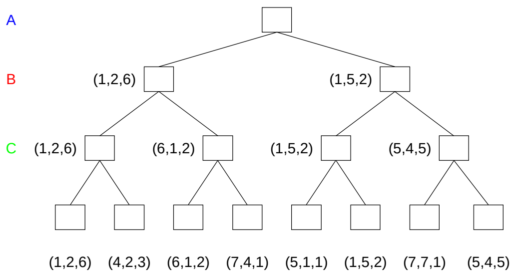
Now have a vector of utilities for players (A,B,C). All players maximize their utilities. Note: In two-player, zero-sum games, we have a single value because the values are always opposite.

Minimax With 3 Players



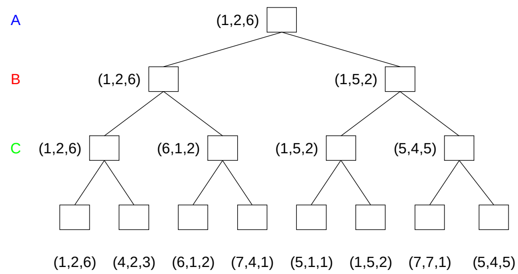
26

Minimax With 3 Players



27

Minimax With 3 Players



28

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

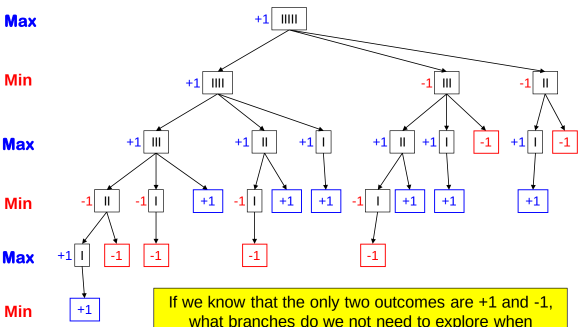
29

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

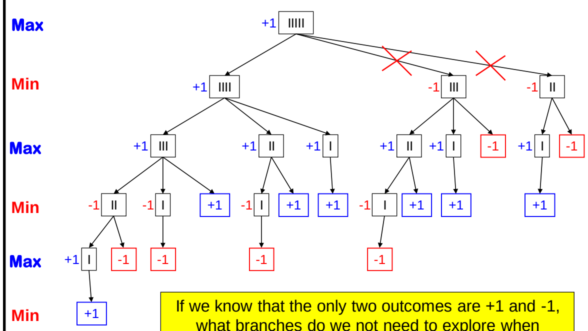
30

Pruning in Nim



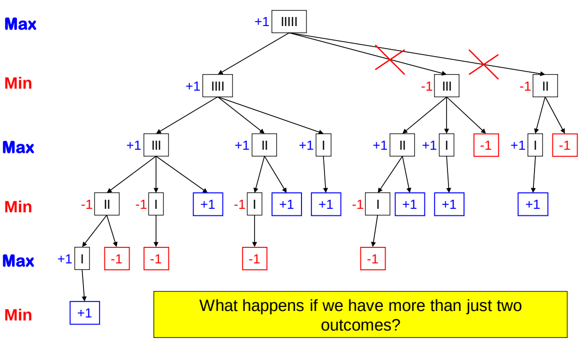
If we know that the only two outcomes are +1 and -1, what branches do we not need to explore when minimax backtracks?

Pruning in Nim



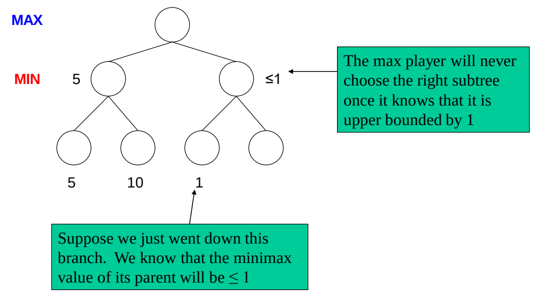
If we know that the only two outcomes are +1 and -1, what branches do we not need to explore when minimax backtracks?

Pruning in Nim

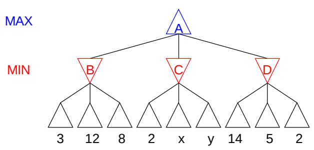


What happens if we have more than just two outcomes?

Pruning Intuition (General Case)



Pruning Example



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 ≤ 2

= 3

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
- β = lowest minimax value of all of the MIN player's choices expanded on current path
- 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 $\geq \beta$

ALPHA-BETA Pseudocode

```
function ALPHA-BETA-SEARCH(state) returns an action
    inputs: state, current state in game
    v ← MAX-VALUE(state, -∞, +∞)
    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 ← -∞
    for a, s in SUCCESSORS(state) do
         v ← MAX(v, MIN-VALUE(s, α, β))
         if v ≥ β then return v
          α ← MAX(α, v)
    return v
```

37

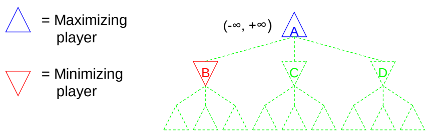
ALPHA-BETA Pseudocode

```
function MIN-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 ← +∞
    for a, s in SUCCESSORS(state) do
         v ← MIN(v, MAX-VALUE(s, α, β))
         if v ≤ α then return v
          β ← MIN(β, v)
    return v
```

38

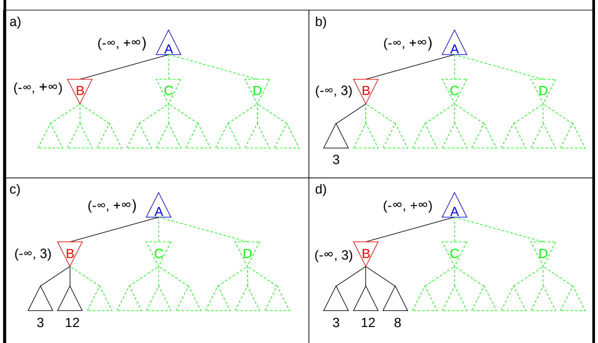
Illustrating the Pseudocode

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

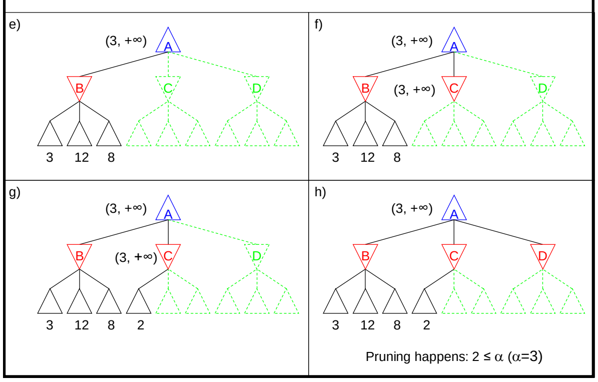


39

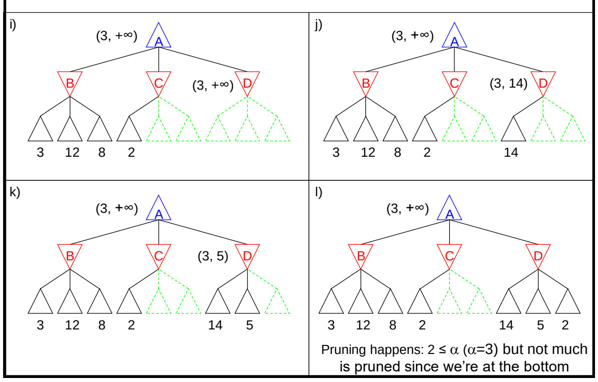
Alpha-Beta Pruning Example



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

43

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)

44

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

45