Lecture 12

Games

Lusi Li

Department of Computer Science

ODU

Reading for This Class:

Chapter 5, Russell and Norvig

Review

- **Last Class**
  - **Constraint Satisfaction Problems**
  - **Backtracking Search Algorithms**

- **This Class**
  - **Games**
  - **Adversarial Search**

- **Next Class**
  - **Pruning a Game Tree**

Games

- Games: multi-agent environment
  - Unpredictability of other agents
  - Possible contingencies in problem solving
  - Cooperative vs. competitive
  - Competitive multi-agent environments give rise to adversarial search a.k.a. games

- Why study games?
  - Fun!
  - They are hard
  - Easy to represent and agents restricted to small number of actions… sometimes!

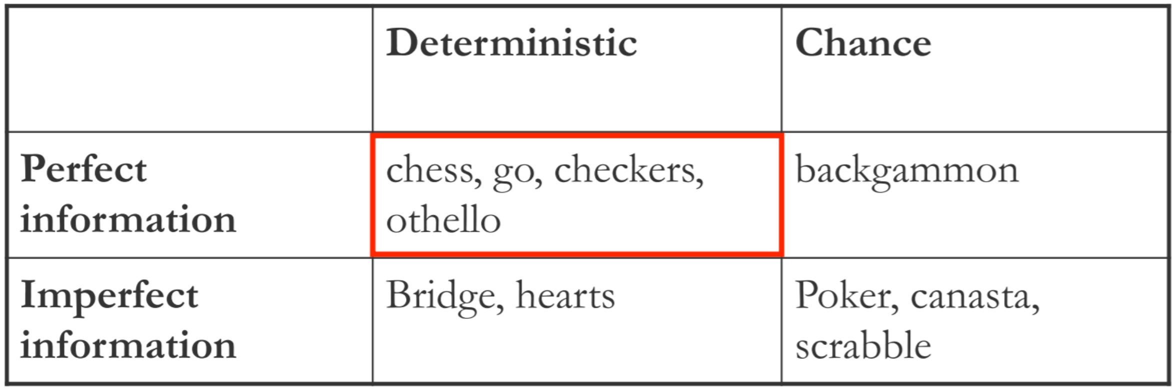
Games vs. Search Problem

- **Search – no adversary**
  - So far: our search problems have assumed agent has complete control of environment
  - Solution is a single path to a goal state
  - Heuristics can find optimal solution
  - Evaluation function: estimate of cost from start to goal through given node
  - Examples: traveling problem, puzzle problem

- **Games – adversary**
  - Assumption is not always reasonable
  - Solution is a strategy which specifies a move for every possible opponent reply.
  - Time limits force approximate solutions
  - Examples: chess, checkers, Othello, backgammon

Types of Games

- Information
  - Perfect Information
    - Fully-observable environment
  - Imperfect Information
    - Unclear environment
- Uncertainty
  - Deterministic Games
    - The outcome is fixed
  - Non-Deterministic Games
    - The outcome is by chance



	Deterministic	Chance
Perfect information	chess, go, checkers, othello	backgammon
Imperfect information	Bridge, hearts	Poker, canasta, scrabble

Zero-sum Games and Collaborate Games

- **Two-player Zero-sum Games**
  - "Fully competitive"
  - Zero-sum means what is good for one player is just as bad for the other. Key insight:
    - how you act depends on how the other agent acts (or how you think they will act)
    - and vice versa (if your opponent is a rational player)
    - no "win-win" outcome

- **Two-player Collaborate Games**
  - "Cooperative"
  - Agents are collaborating to achieve the highest score
  - "win-win" outcome

Two-Person Games

- Games with two opposing players
  - often called MAX and MIN
  - usually MAX moves first, then they take turns
  - a sequence of two moves, one move by each player, is called a *ply*
- MAX wants a strategy to find a winning state
  - no matter what MIN does
- MIN does the same
  - or at least tries to prevent MAX from winning
- Perfect information
  - both players know the full state of the environment

Game Setup

- Two players: MAX and MIN
- MAX moves first and they take turns until the game is over.
- A game can be defined as a search problem:
  - `initial state` s_0: how the game is set up at the start
  - `Player(s)`: which player has the move in state s
  - `Action(s)`: set of legal moves in state s
  - `Result(s, a)`: the state resulting from action a in state s
  - `Terminal-Test(s)`: true if game is over (terminal state) otherwise false
  - `Utility(s,p)`: a numeric value of a terminal state s for a player p, e.g., win (+1), lose (-1) and draw (0) in chess.

- The s_0, `Action(s)` and `Result(s, a)` define the game tree, where the nodes are game states and the edges are moves.

- Players use game tree to determine next move.

Example of Tic-Tac-Toe Game Tree

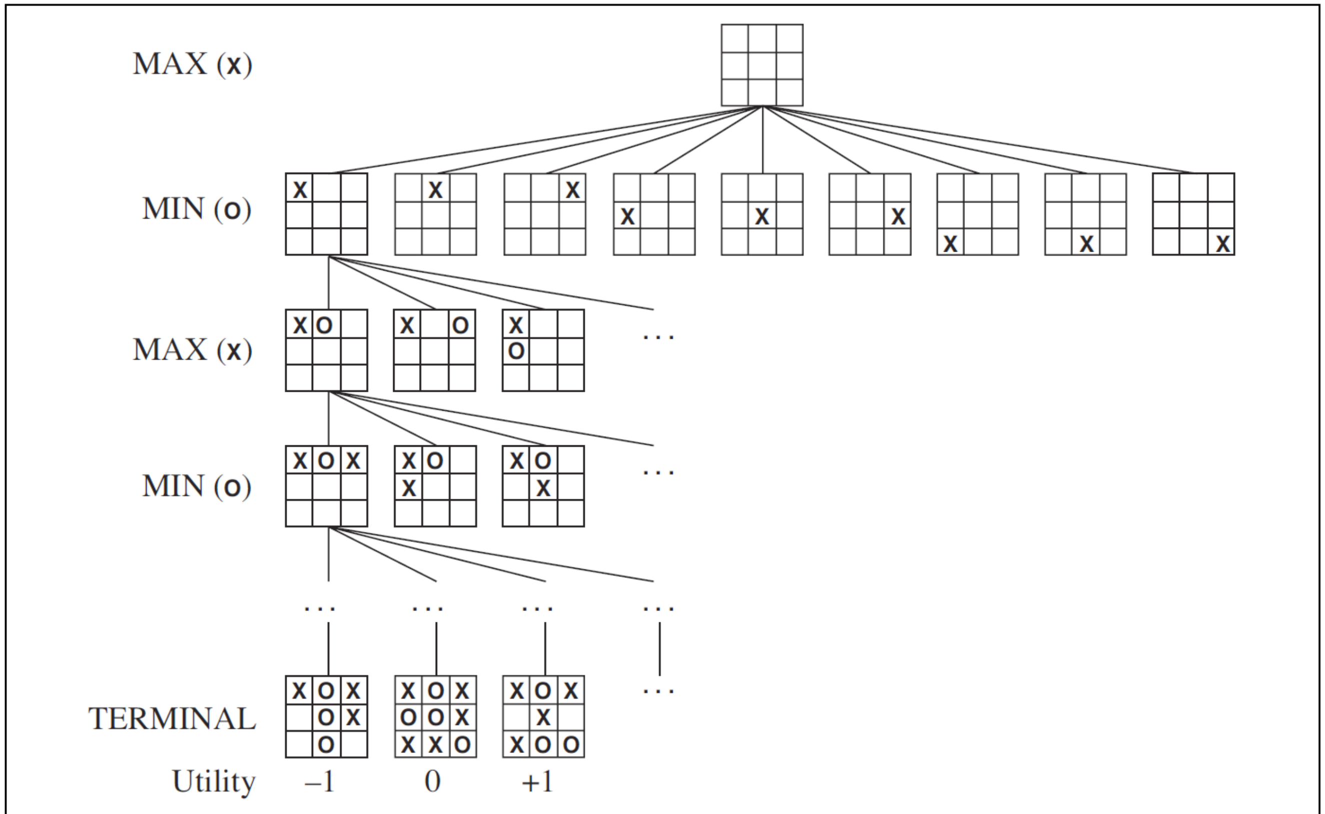


Figure 5.1 A (partial) game tree for the game of tic-tac-toe. The top node is the initial state, and MAX moves first, placing an X in an empty square. We show part of the tree, giving alternating moves by MIN (O) and MAX (X), until we eventually reach terminal states, which can be assigned utilities according to the rules of the game.

- Assume depth-first generation of game tree
- Game tree looks like a search tree and levels reflect the alternating moves.
- The game tree is relatively small, fewer than 9! = 362, 880 terminal nodes.
- MAX does not decide the next move alone and must have a strategy.
- What is a reasonable strategy?

Optimal Strategy

- Find the best strategy for MAX assuming an infallible MIN opponent.

- Assumption: both players play optimally.

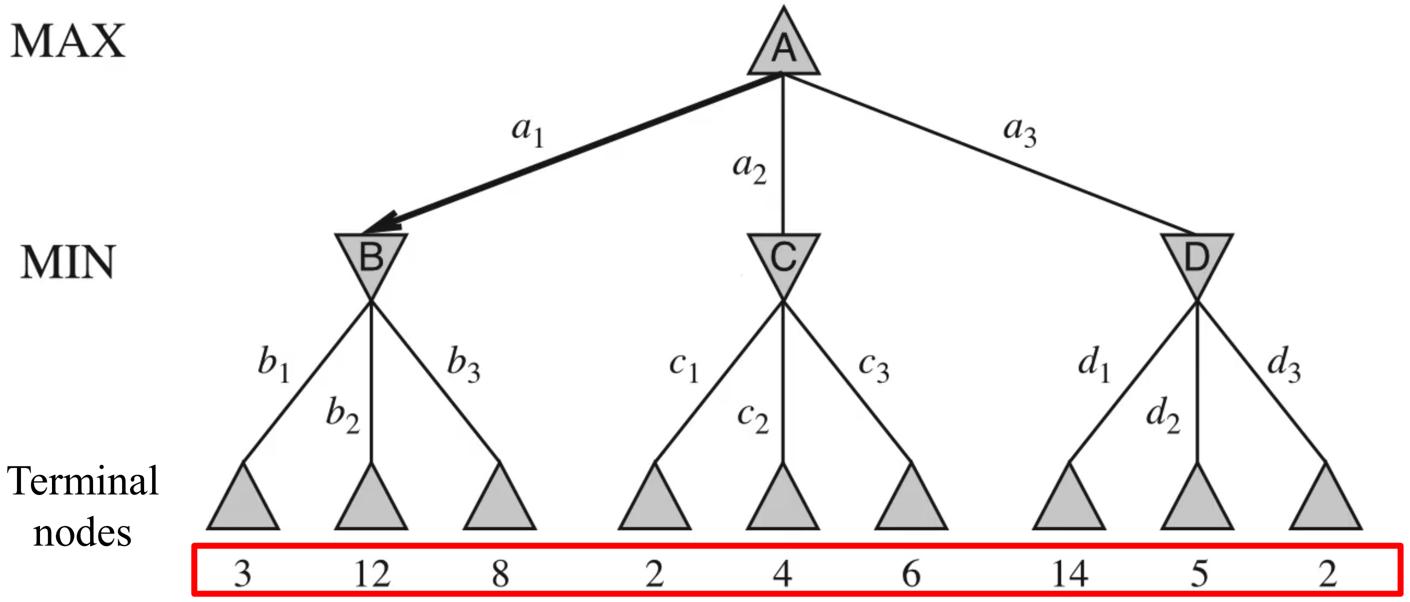
- (1) Generate full game tree (all leaves are terminals) using DFS
  - root is start state, edges are possible moves, etc.
  - label terminal nodes with utilities
- (2) Given the game tree, the optimal strategy can be determined by using the minimax value of each node MiniMax(s).
  - Explore the tree to reach terminal states
  - Evaluate the Utility of the terminal states, i.e., minimax values of the terminal states
  - Propagate minimax values up the game tree

MiniMax Value

- Propagate minimax values up the game tree
  - Start at the leaves
  - Assign value to the parent node as follows:
    - If MIN node's move, propagate up minimum value of its successors
    - If MAX node's move, propagate up maximum value of its successors
- The MiniMax function considers all possible ways the game can go and returns the value of the state

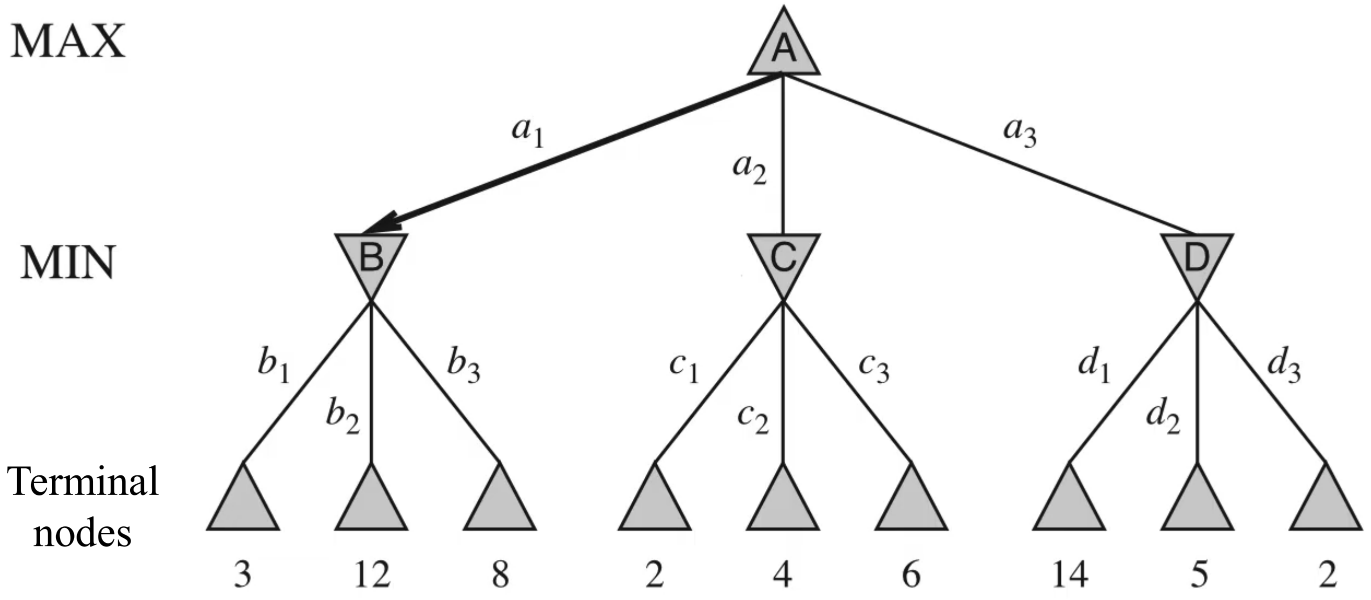
```
MiniMax(s) =

if Terminal-Test(s) then Utility(s)
if Player(s) = MAX then
     max of MiniMax(Result(s, a)) for a in Actions(s)
if Player(s) = MIN then
     min of MiniMax(Result(s, a)) for a in Actions(s)
```

MiniMax Value Example: game tree



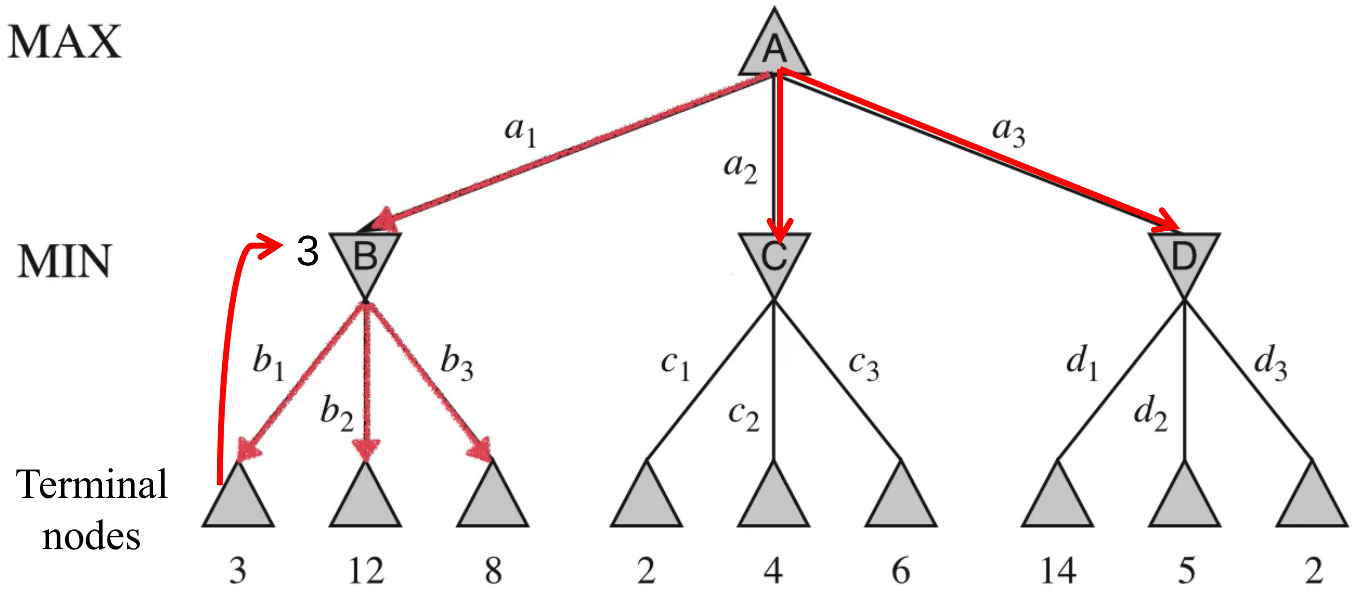
- **This game tree is one move deep, consisting of two plies.**
- **Definition: *ply* = each move by one player of a two-player game**
- **ONLY the terminal nodes have utilities. But we can compute a “utility” for the non-terminal states, by assuming both players always play their best move.**

MiniMax Value Example



```
MiniMax(s) =

if Terminal-Test(s) then Utility(s)
if Player(s) = MAX then
     max of MiniMax(Result(s, a)) for a in Actions(s)
if Player(s) = MIN then
     min of MiniMax(Result(s, a)) for a in Actions(s)
```

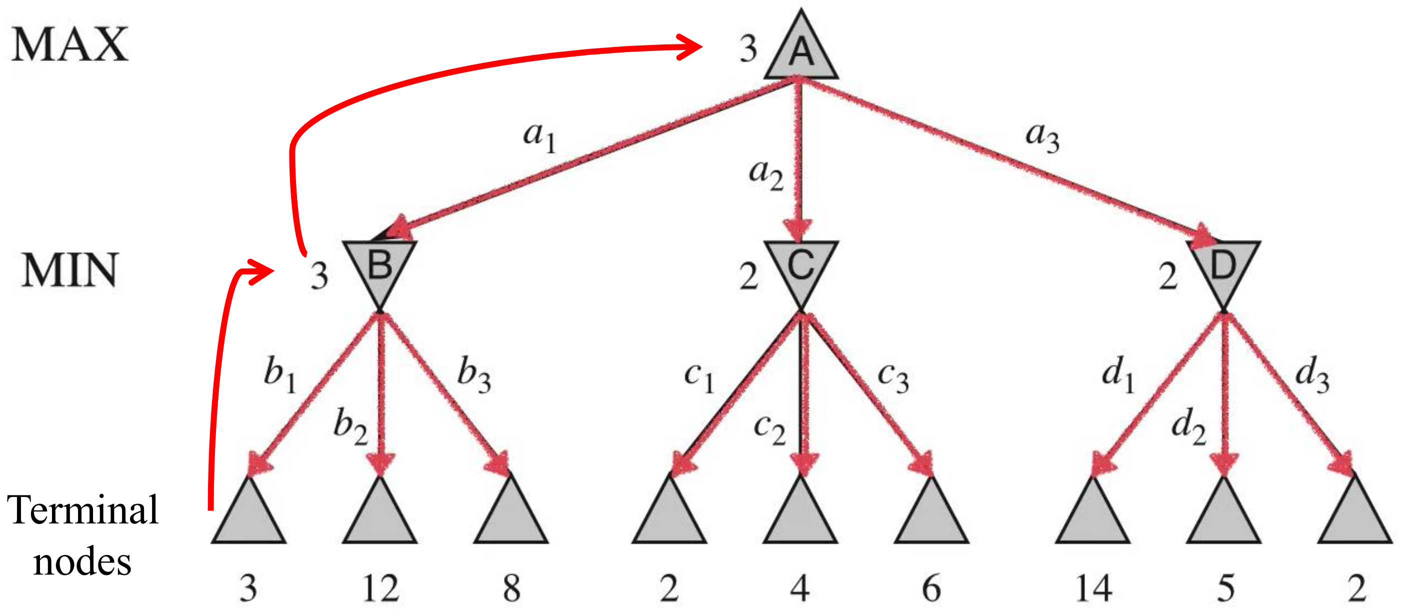
MiniMax Value Example



The minimax value at a MIN node is the minimum of backed-up values, because your opponent will do what's best for them (and worst for you).

```
MiniMax(s) =

if Terminal-Test(s) then Utility(s)
if Player(s) = MAX then
     max of MiniMax(Result(s, a)) for a in Actions(s)
if Player(s) = MIN then
     min of MiniMax(Result(s, a)) for a in Actions(s)
```

MiniMax Value Example

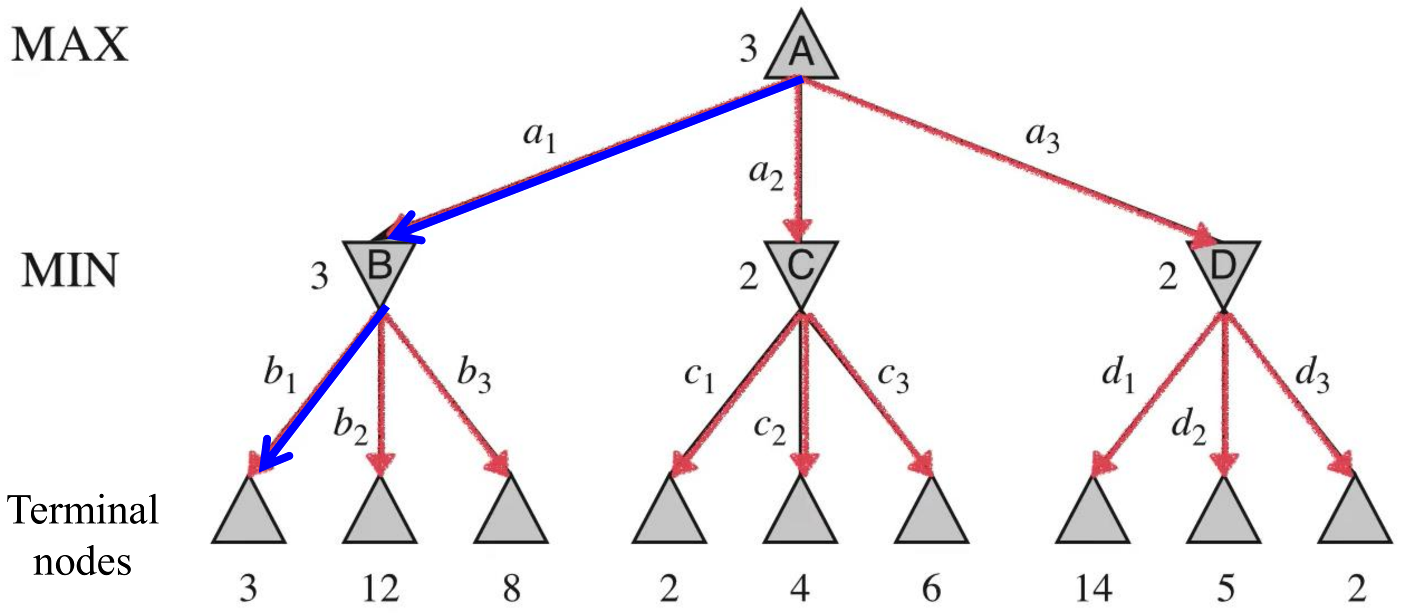


Propagate values up the tree.

Minimax maximizes the worst-case outcome for a MAX node.

```
MiniMax(s) =

if Terminal-Test(s) then Utility(s)
if Player(s) = MAX then
     max of MiniMax(Result(s, a)) for a in Actions(s)
if Player(s) = MIN then
     min of MiniMax(Result(s, a)) for a in Actions(s)
```

MiniMax Value Example

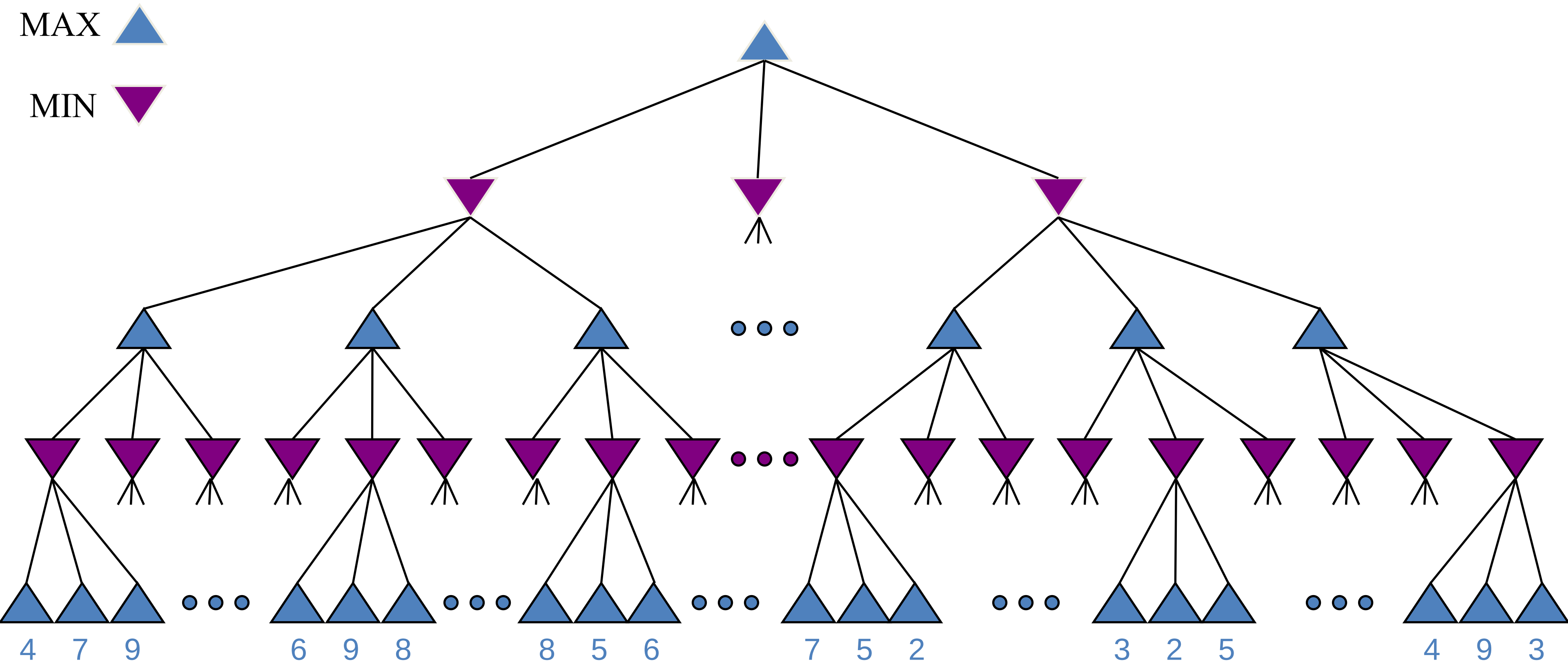


The best move for MAX is a_1

The best move for MIN is b_1

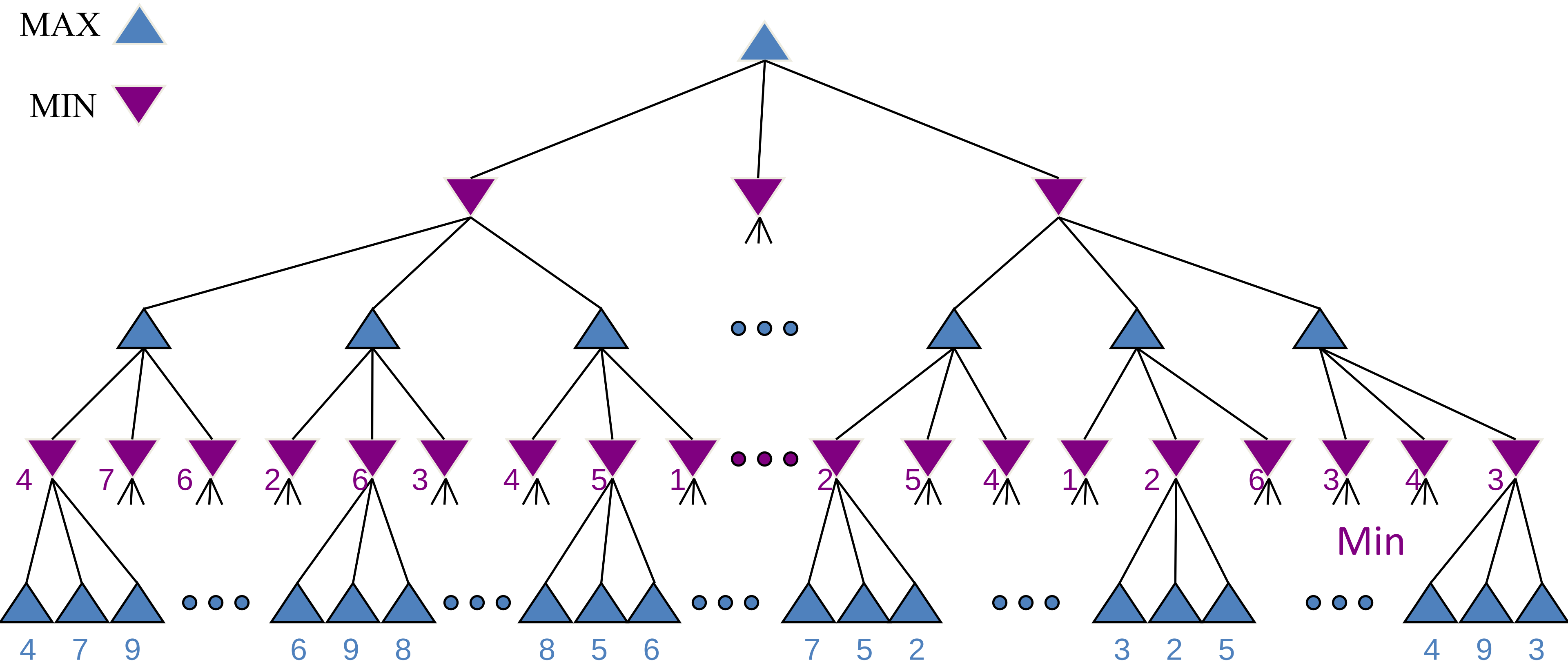
```
MiniMax(s) =

if Terminal-Test(s) then Utility(s)
if Player(s) = MAX then
     max of MiniMax(Result(s, a)) for a in Actions(s)
if Player(s) = MIN then
     min of MiniMax(Result(s, a)) for a in Actions(s)
```

MiniMax Example



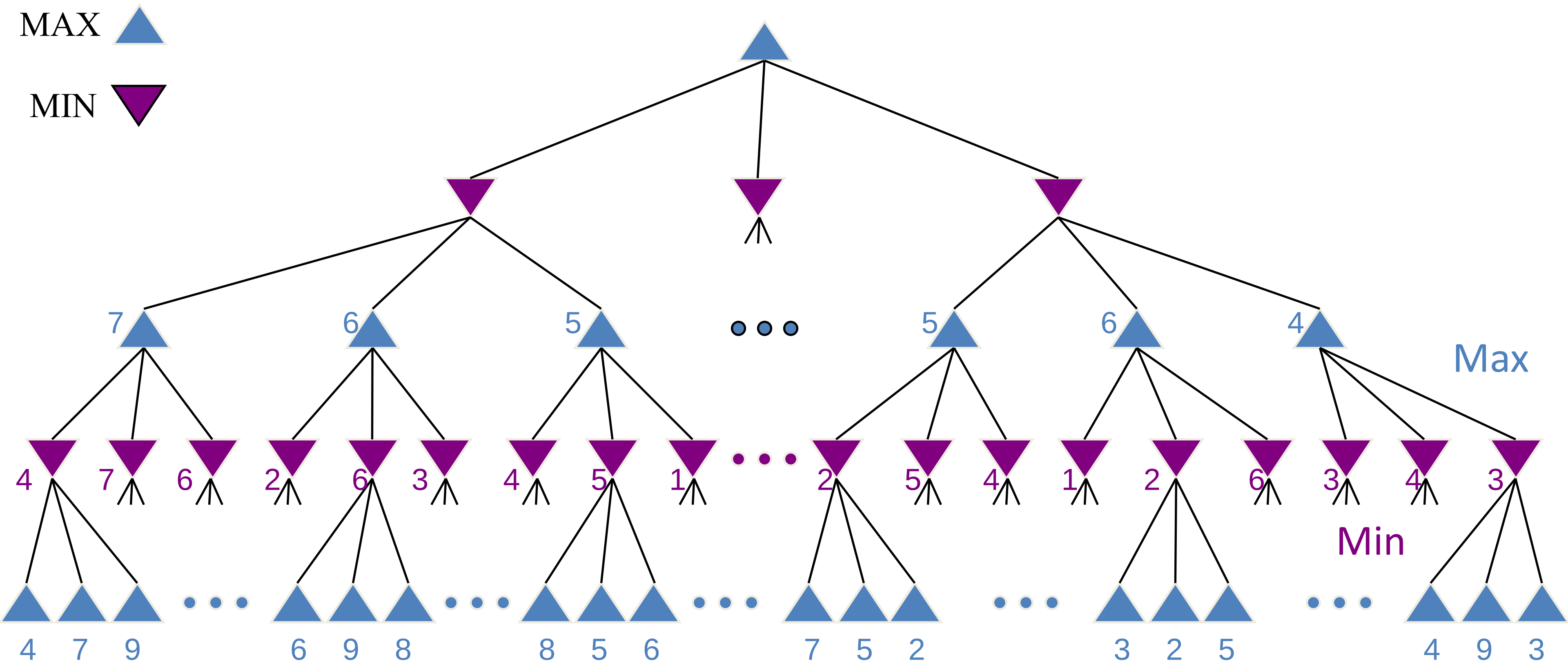
terminal nodes: values calculated from the utility function

MiniMax Example

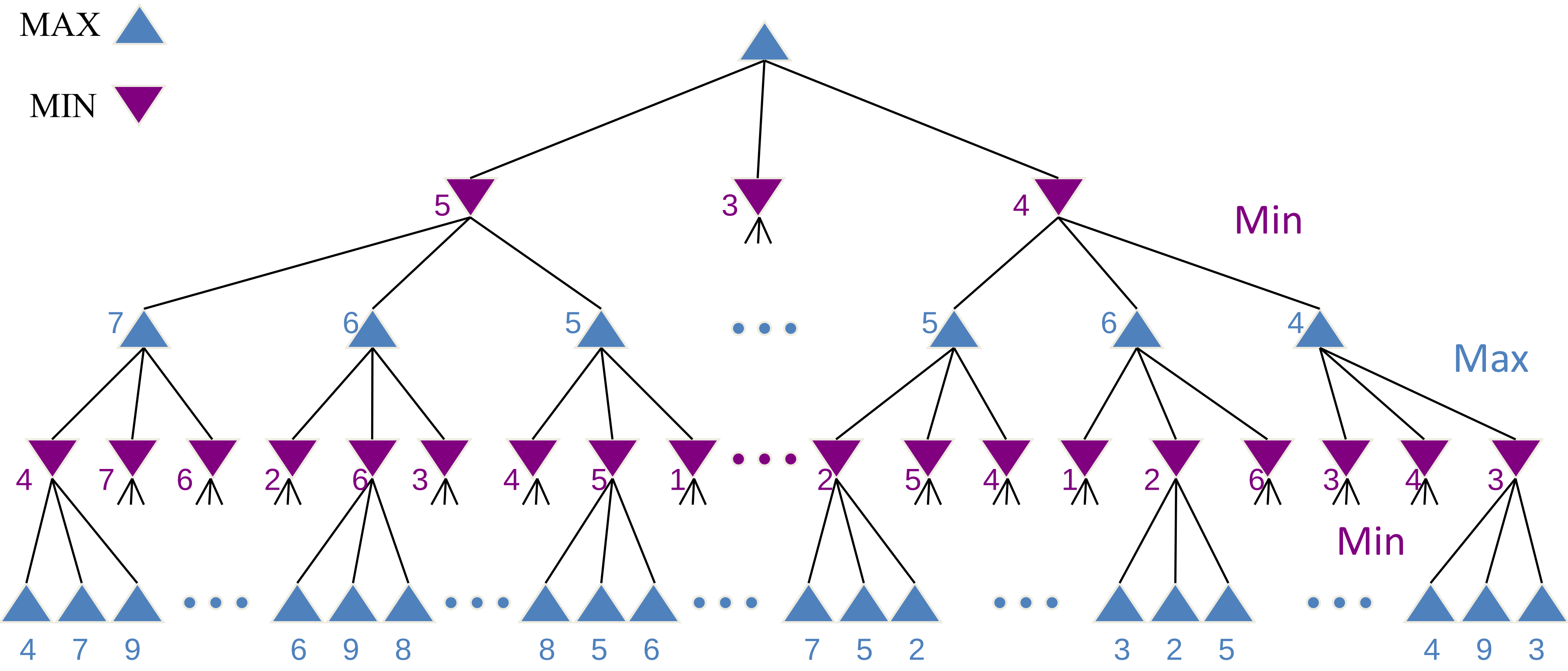


other nodes: values calculated via minimax algorithm

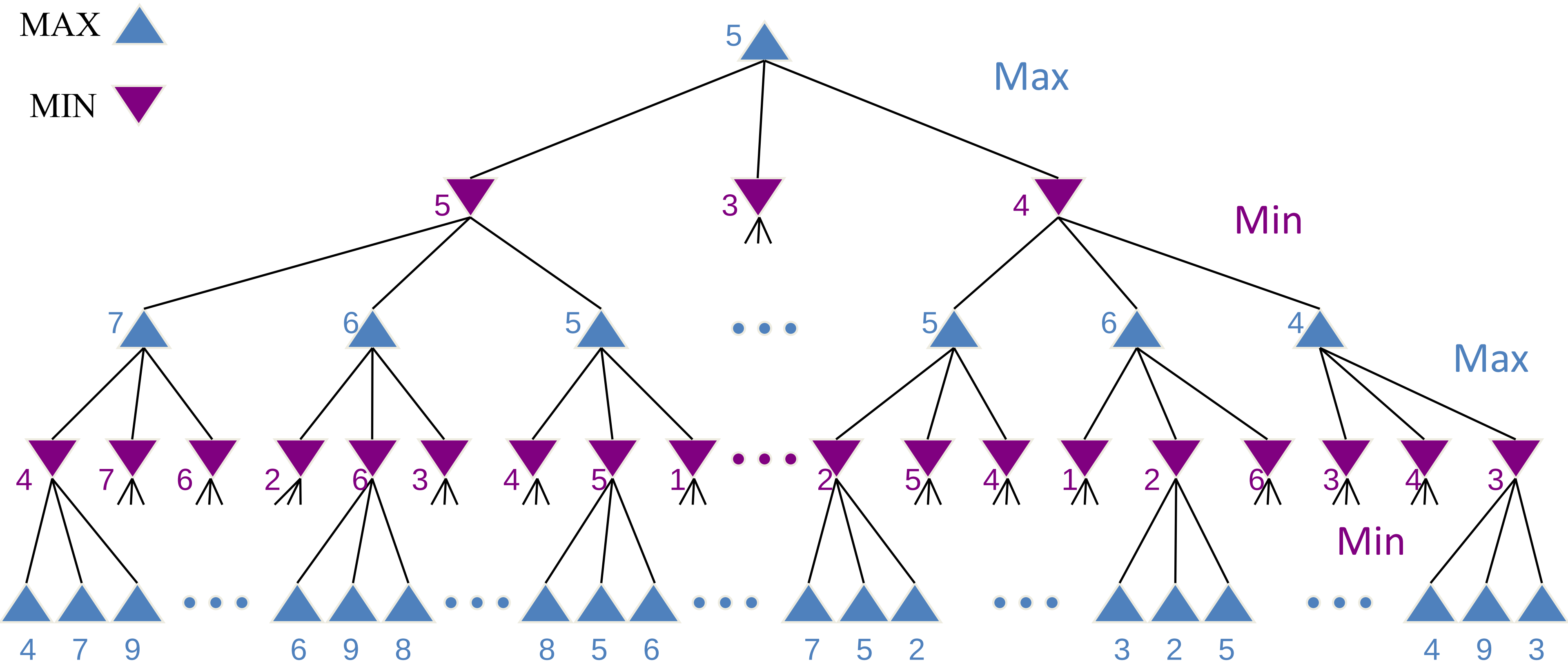
MiniMax Example



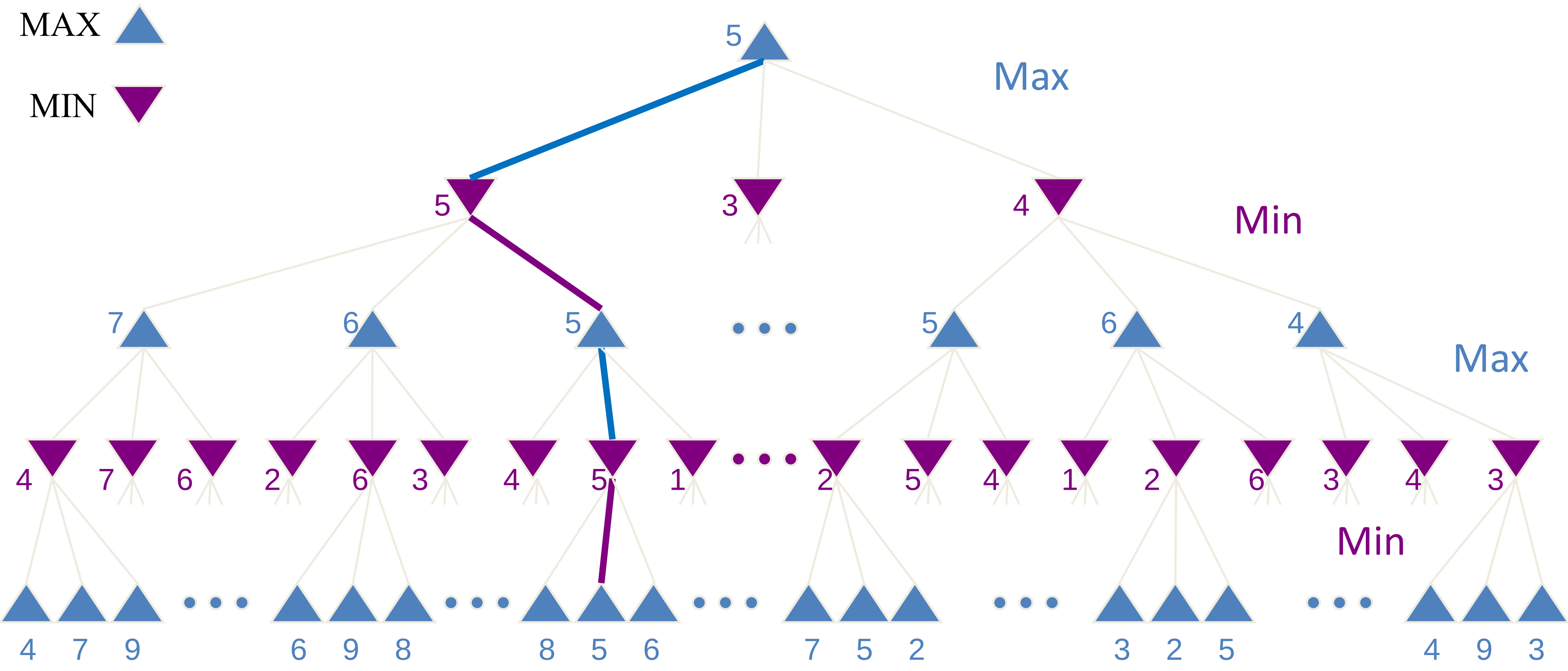
MiniMax Example



MiniMax Example



MiniMax Example



moves by Max and countermoves by Min

Minimax Algorithm

```
function MINIMAX(N) is
    begin
    if N is a leaf
    then
          return the estimated score of this leaf
    else
          Let N1, N2, .., Nm be the successors of N;
          if N is a Min node
          then
                    return min{MINIMAX(N1), .., MINIMAX(Nm)}
          else
                    return max{MINIMAX(N1), .., MINIMAX(Nm)}
    end if
end function
```

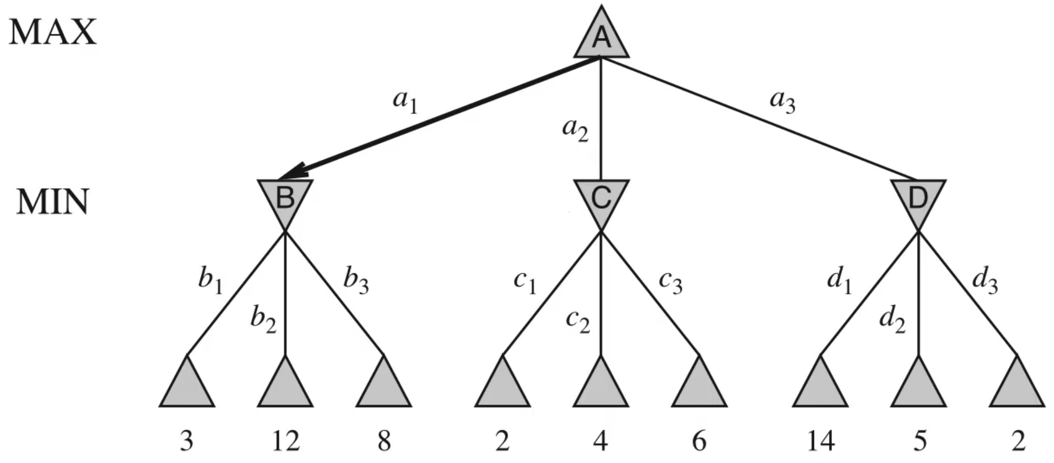
Minimax Algorithm for a MAX node

```
function MINIMAX-DECISION(state) returns an action
    return $\arg\max_{a \in \text{ACTIONS}(s)}$ MIN-VALUE(RESULT(state, a))
```

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

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

Minimax Algorithm for a MAX node



```
function MINIMAX-DECISION(state) returns an action
    return arg max_{a ∈ ACTIONS(s)} MIN-VALUE(RESULT(state, a))
```

```
function MAX-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← −∞
    for each a in ACTIONS(state) do
        v ← MAX(v, MIN-VALUE(RESULT(s, a)))
    return v
```

```
function MIN-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← ∞
    for each a in ACTIONS(state) do
        v ← MIN(v, MAX-VALUE(RESULT(s, a)))
    return v
```

MiniMax Properties

- Assume all terminal states are at depth m and there are b possible moves at each step
- Minimax explores tree using DFS.
  - recursive implementation
- time complexity is $O(b^m)$
  - exponential in the number of moves
- space complexity is $O(bm)$
  - that generates all actions at once
  - where b is the branching factor, m the maximum depth of the search tree
  - Or $O(m)$ that generates actions one at a time
- Completeness: Yes, if tree is finite
- Optimality: Yes, if MIN is an optimal opponent

Why so hard to be a perfect player?

- **Search Complexity**
  - **Time Complexity (Main Issue)**
  - **Space Complexity**

- **In many cases, we can only come to some level of the game tree**
  - **Example: Chess with game tree of about 35^{100} nodes**
  - **It is not feasible to search all states.**

MiniMax Observations

- **The values of some of the leaf nodes are irrelevant for decisions at the next level**
- **This also holds for decisions at higher levels**
- **As a consequence, under certain circumstances, some parts of the tree can be disregarded**
  - **it is possible to still make an optimal decision without considering those parts**

Summary

- **Games**
  - Categories
    - Perfect Information vs. Imperfect Information
    - Zero-sum game and collaboration game

- **Minimax algorithm**
  - How to find the minimax value
  - The minimax algorithm

What I want you to do

- Review Chapter 5
- Work on your Assignment 2