Artificial Intelligence

Kushal Shah @ Sitare

Adversarial Search

- Game Playing [Section 5.1]
- Understand and applying Mini-max [Section 5.2.1]
- Reducing search space with Alpha-Beta Pruning [Section 5.3]

Given an initial state of a 8-puzzle problem and final state to be reached-

2	8	3
1	6	4
7		5

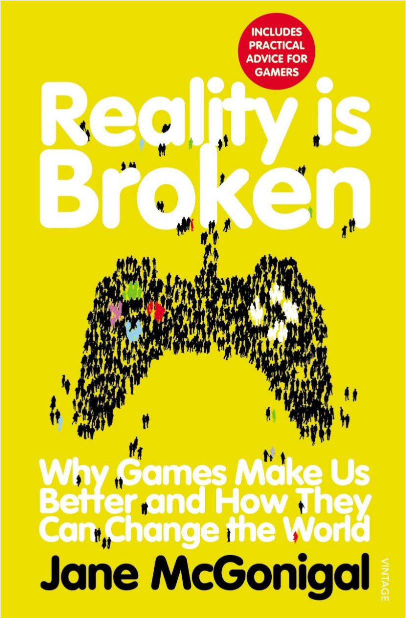
Initial State

1	2	3
8		4
7	6	5

Final State

Find the most cost-effective path to reach the final state from initial state using A* Algorithm.

Games in AI



- Any multi-agent environment is generally a 'game'

- Typical Game components:
  - Well-defined Goal
  - Well-defined Rules
  - Feedback System
  - Voluntary Participation

- In classical AI, the most common games are of a rather specialized kind
  - Deterministic and Fully observable environment
  - Two players taking turns
  - Zero-sum 'adversarial' games of perfect information (eg. chess)

Games in AI

- Games are interesting in AI because they are too hard to solve algorithmically
    - Humans are so far much better at games in general

- Chess has an average branching factor of about 35, and there are about 50 moves by each player
    - So the search tree has about 35^{100} or 10^{154} nodes
    - The search graph has about 10^{40} distinct nodes

- Games require the ability to make lot of good but non-optimal decisions

- Games also penalize inefficiency severely
    - Take too much time and you lose the game!

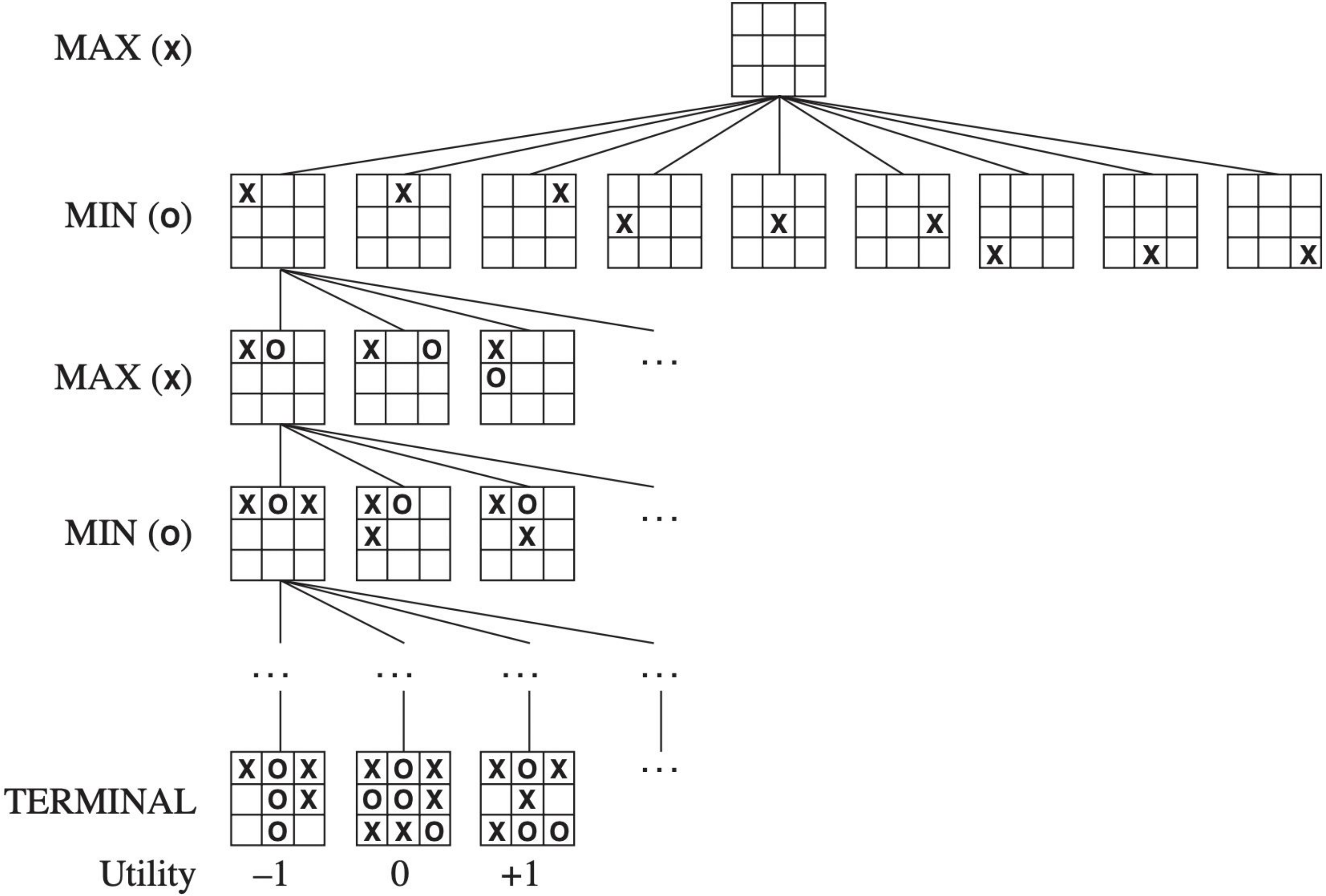


Figure 5.1 A (partial) game tree for the game of tic-tac-toe. The top node is the initial

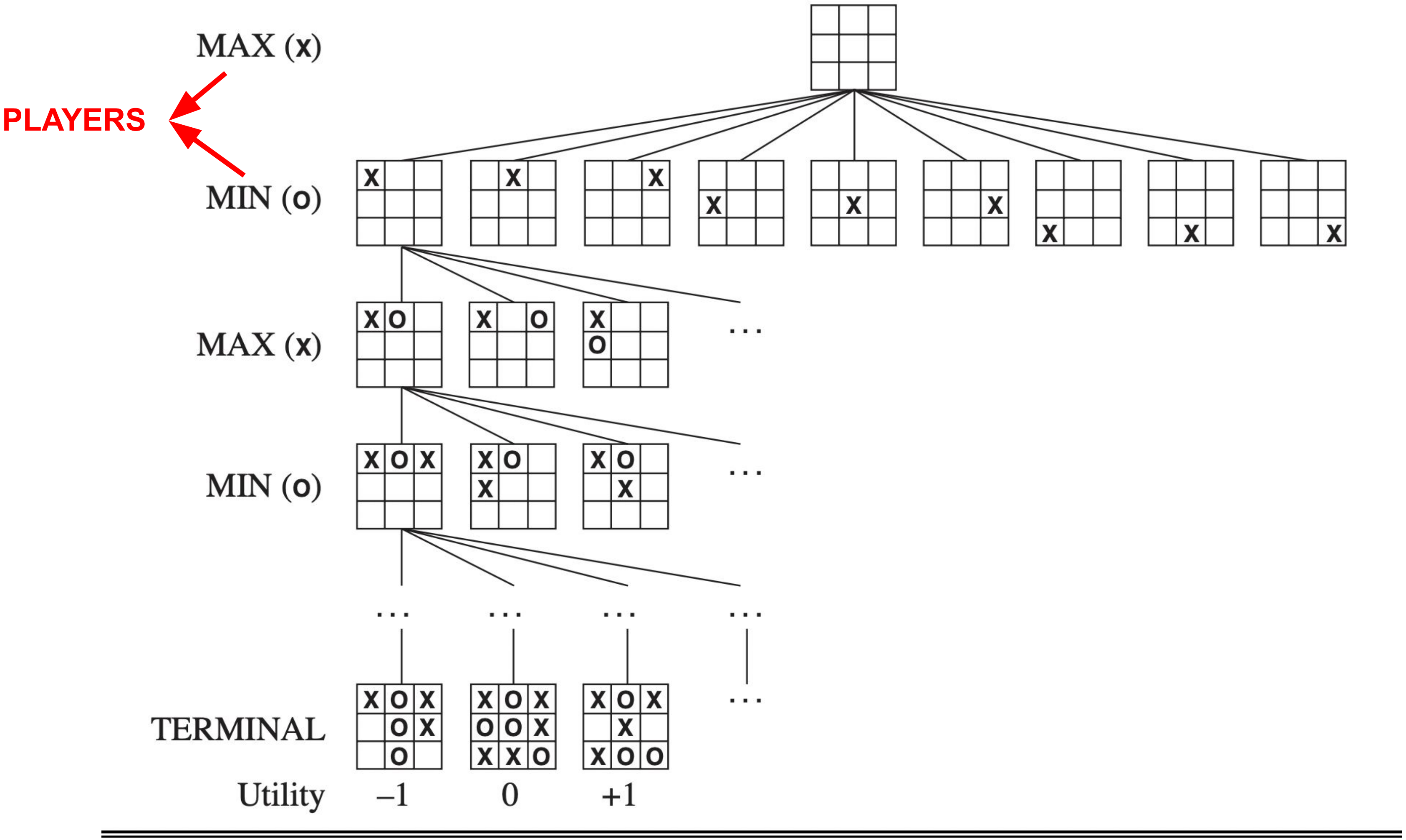


Figure 5.1 A (partial) game tree for the game of tic-tac-toe. The top node is the initial

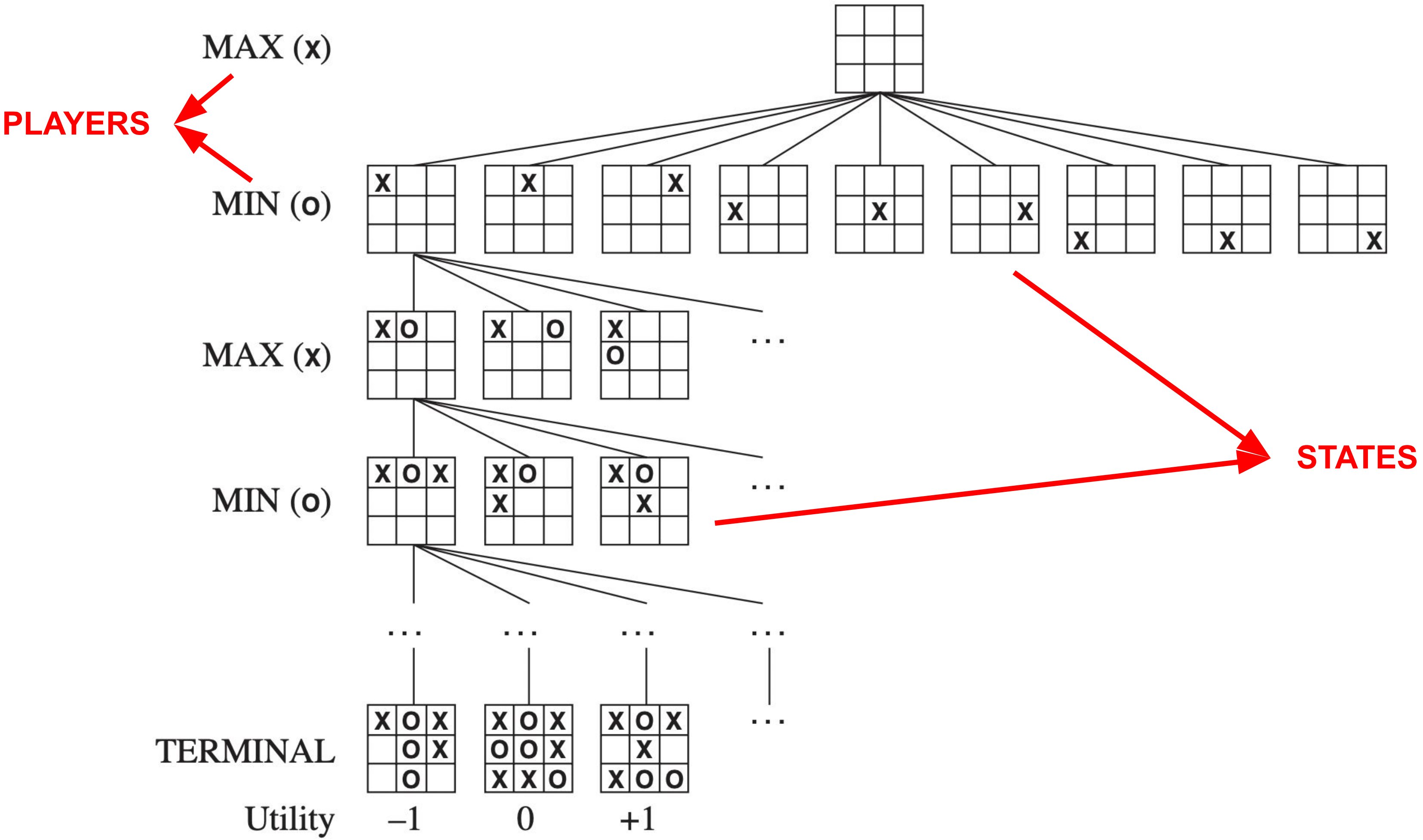


Figure 5.1 A (partial) game tree for the game of tic-tac-toe. The top node is the initial

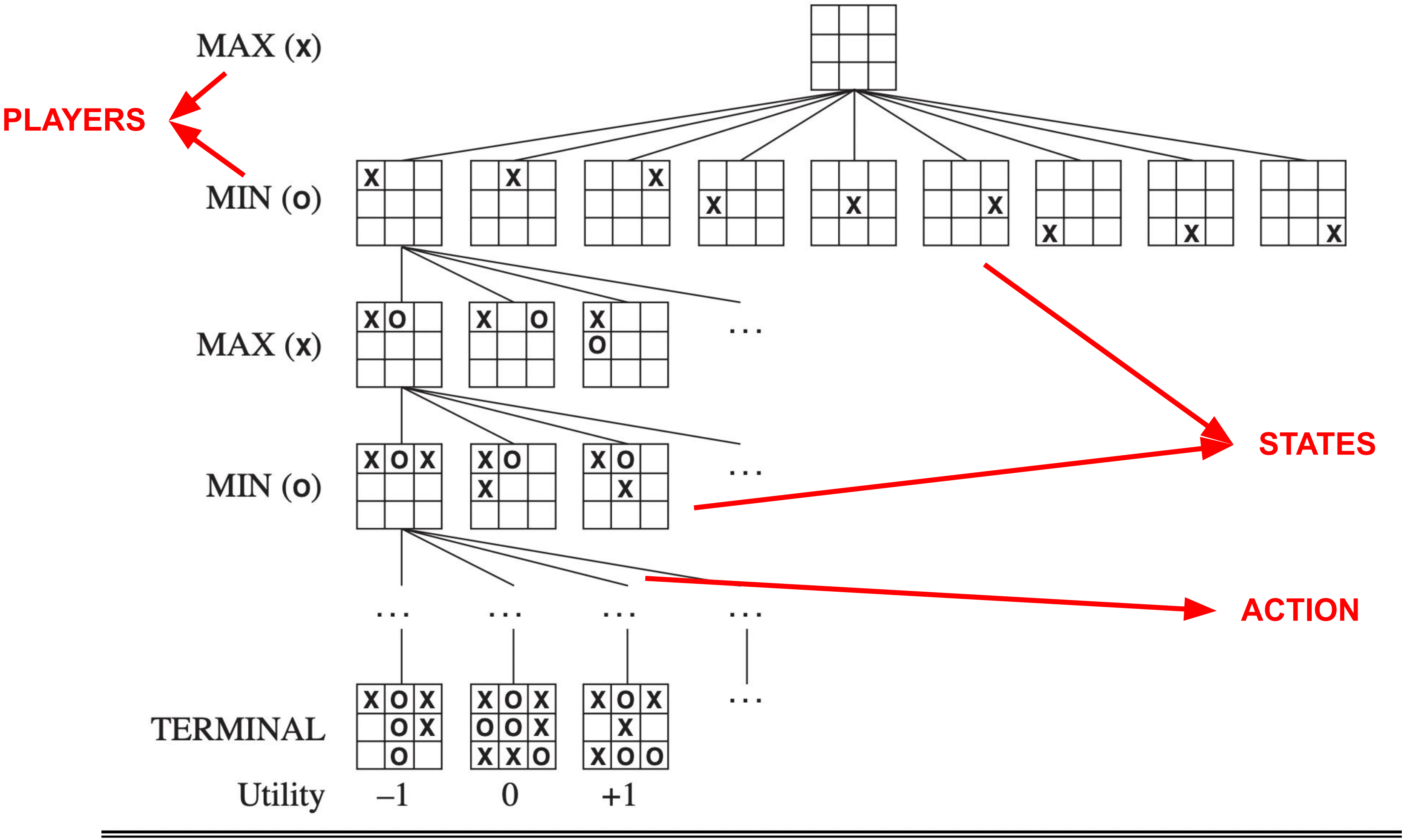


Figure 5.1 A (partial) game tree for the game of tic-tac-toe. The top node is the initial

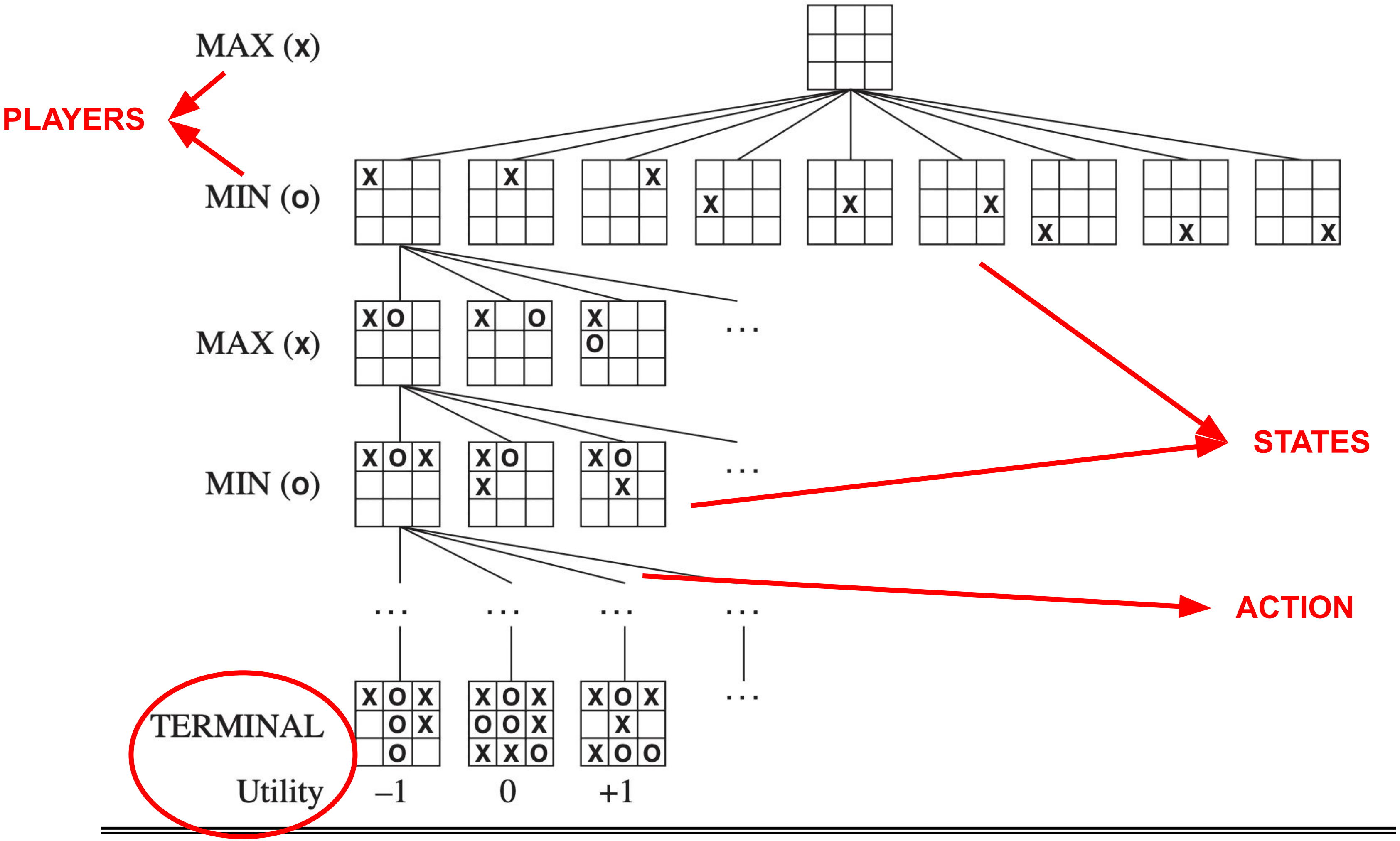


Figure 5.1 A (partial) game tree for the game of tic-tac-toe. The top node is the initial

Definition of a AI game

- S_0: The initial state, which specifies how the game is set up at the start.

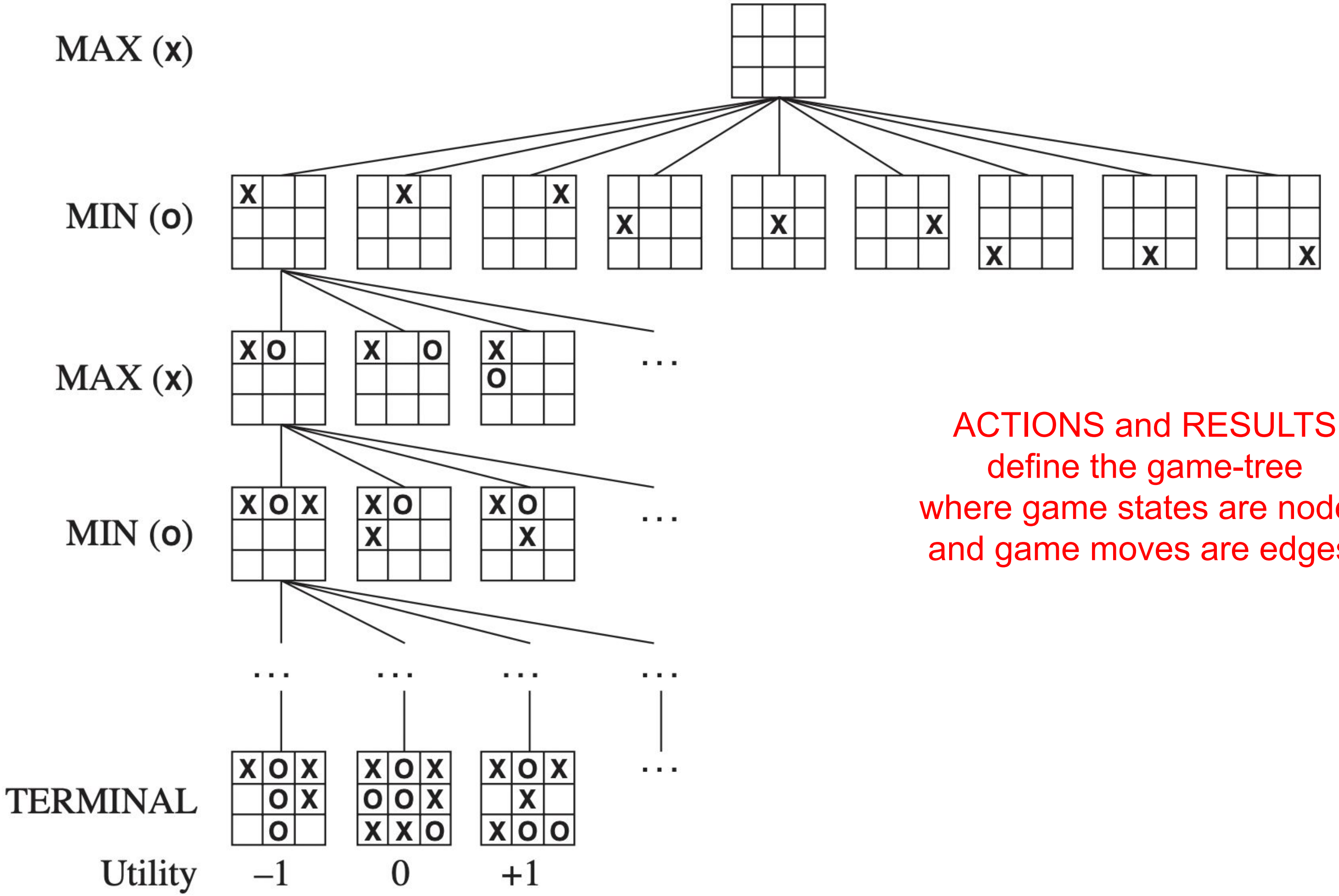
- PLAYER(s): Defines which player has the move in a state.

- ACTIONS(s): Returns the set of legal moves in a state.
- RESULT(s, a): The transition model, which defines the result of a move.

- TERMINAL-TEST(s): A terminal test, which is true when the game is over and false otherwise
    - States where the game has ended are called terminal states.

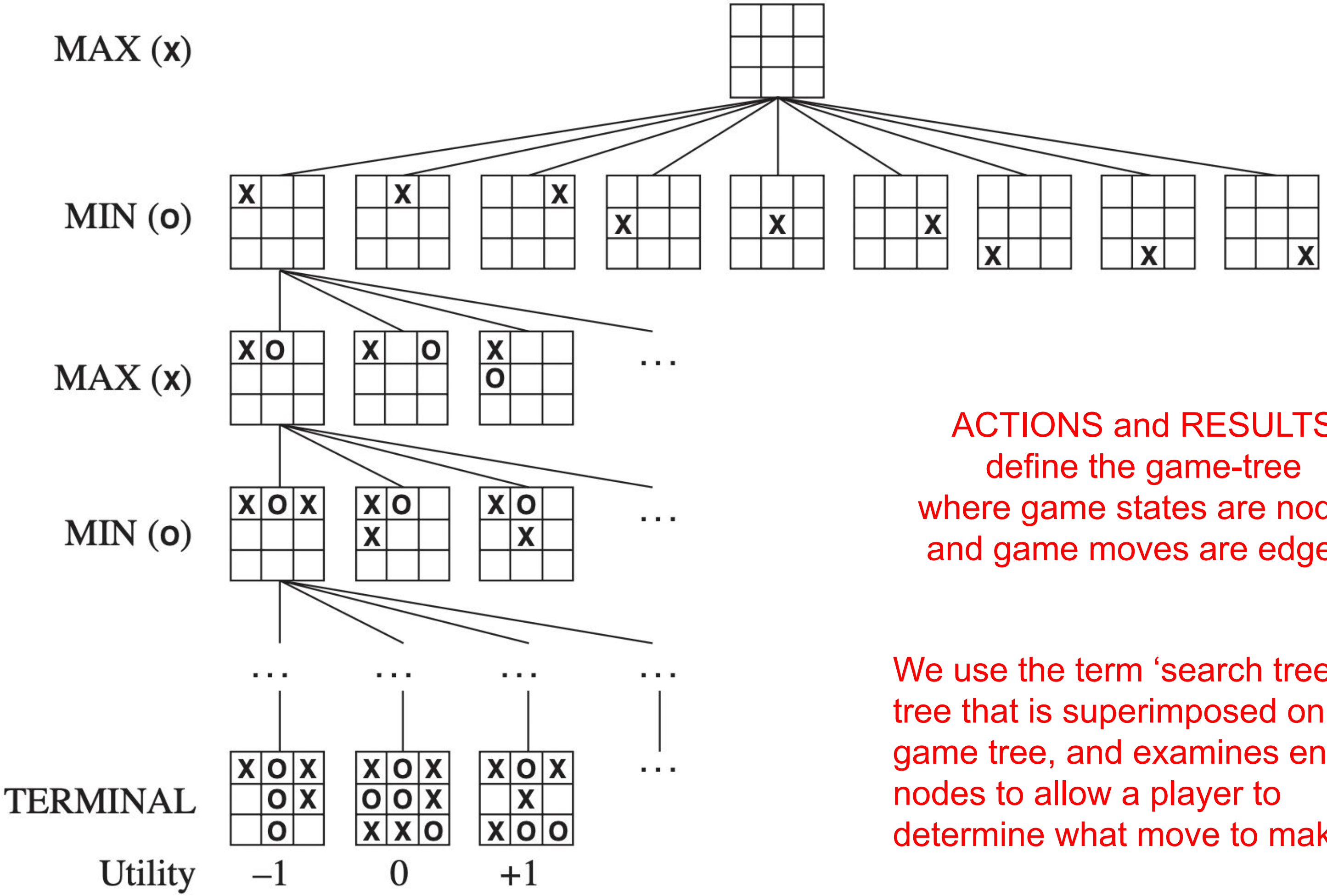
- UTILITY(s, p): A utility function (also called an objective function or payoff function), defines the final numeric value for a game that ends in terminal state s for a player p.
    - In chess, the outcome is a win, loss, or draw, with values +1, 0, or ½
    - The payoffs in backgammon range from 0 to +192

- In a zero-sum game, the total payoff to all players is the same for every instance of the game



ACTIONS and RESULTS
define the game-tree
where game states are nodes
and game moves are edges.

Figure 5.1 A (partial) game tree for the game of tic-tac-toe. The top node is the initial



ACTIONS and RESULTS
define the game-tree
where game states are nodes
and game moves are edges.

We use the term 'search tree' for a tree that is superimposed on the full game tree, and examines enough nodes to allow a player to determine what move to make.

Figure 5.1 A (partial) game tree for the game of tic-tac-toe. The top node is the initial

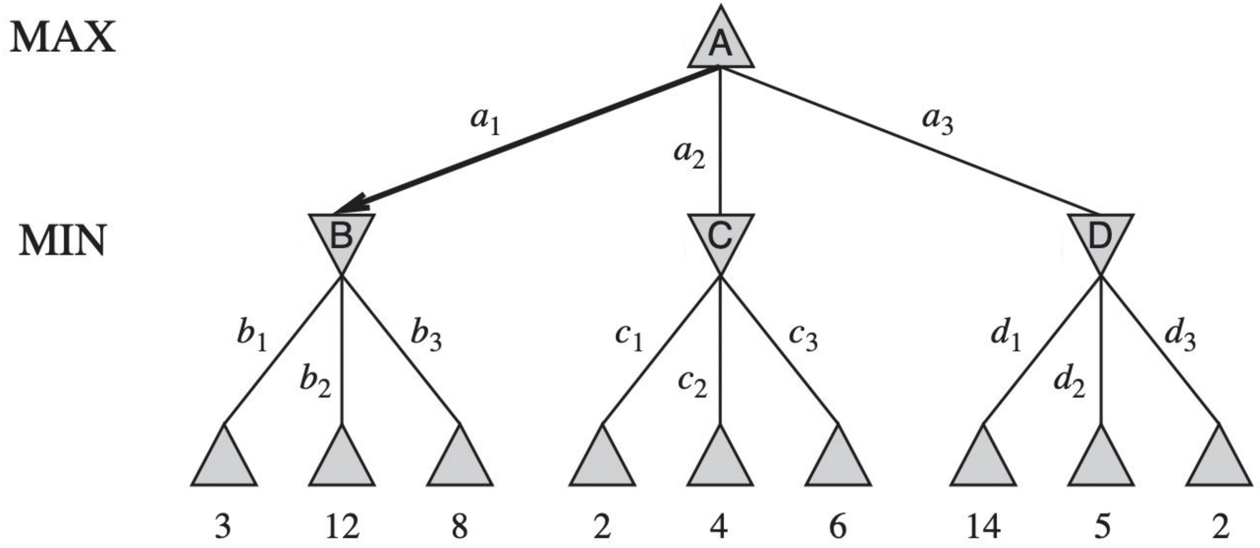
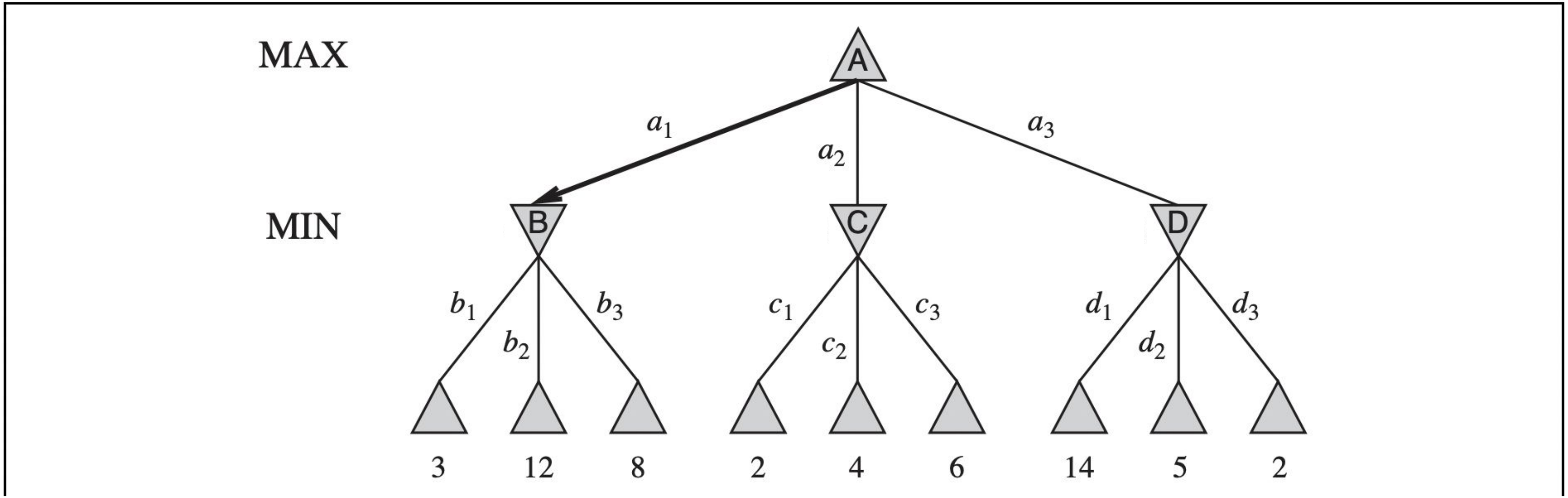
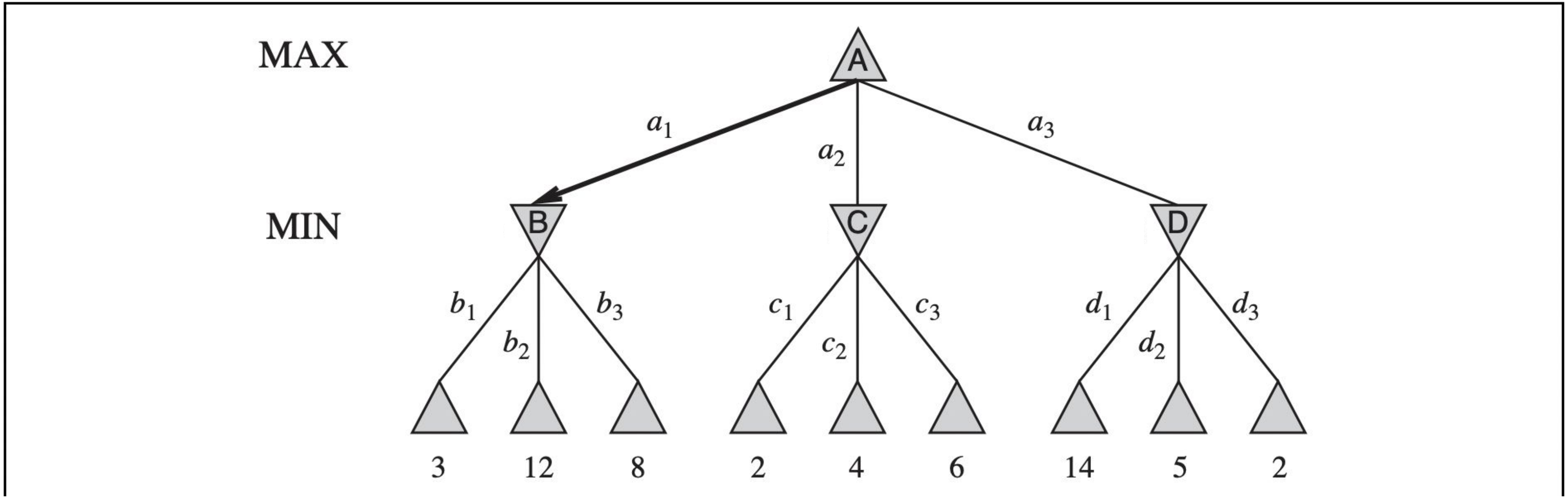


Figure 5.2 A two-ply game tree. The △ nodes are "MAX nodes," in which it is MAX's turn to move, and the ▽ nodes are "MIN nodes." The terminal nodes show the utility values for MAX; the other nodes are labeled with their minimax values.

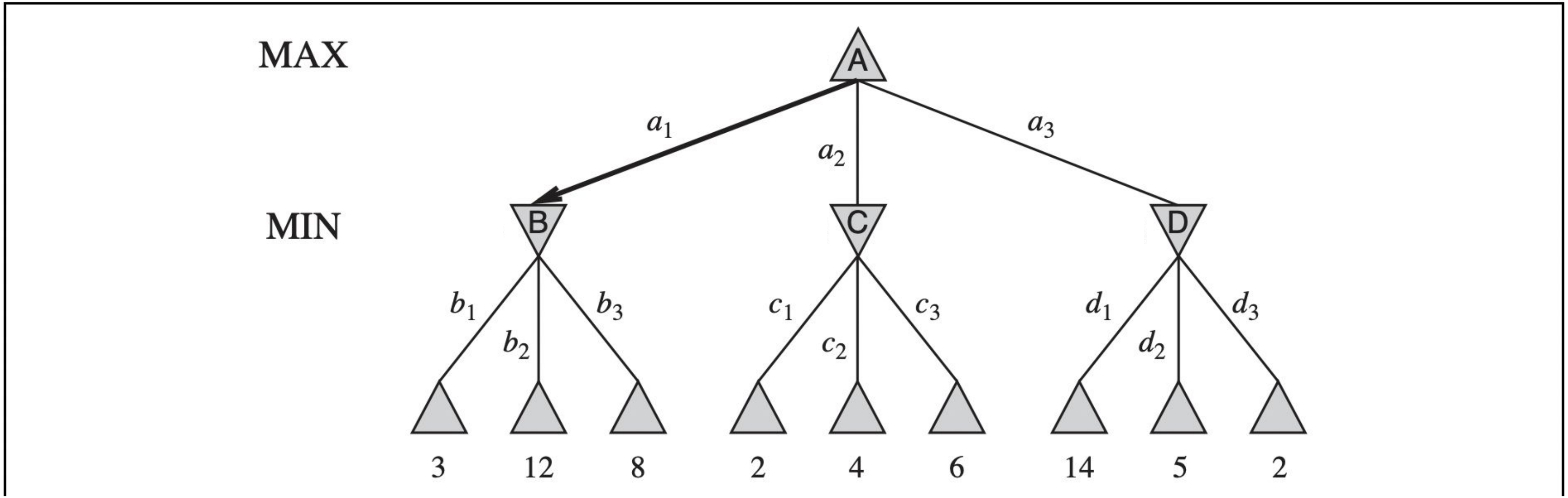


MiniMax Function



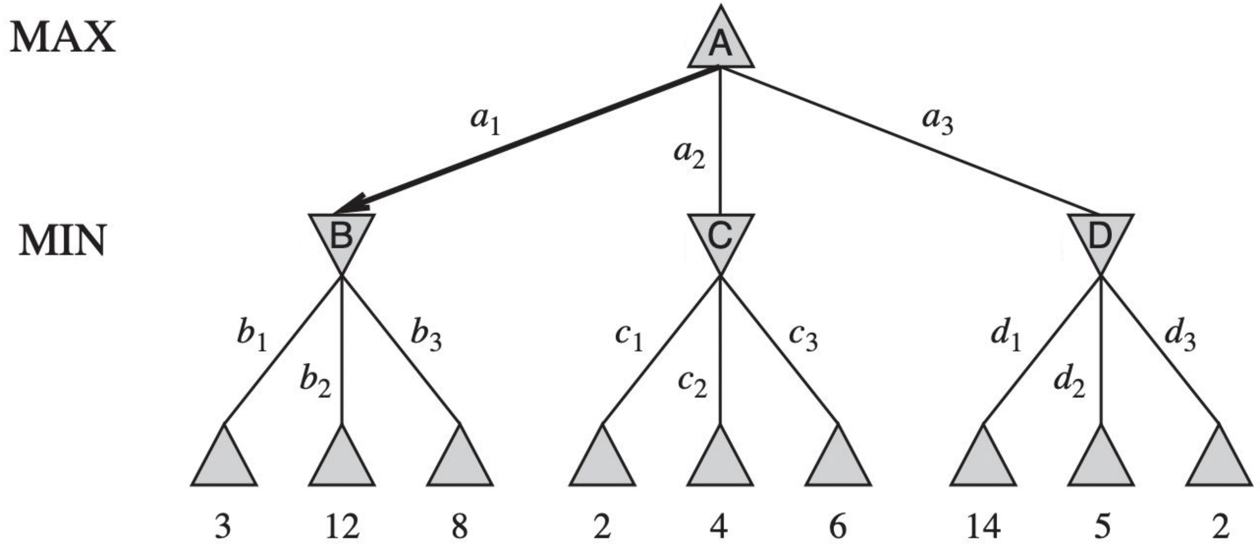
MiniMax Function

- The minimax value of a node is the utility (for MAX) of being in the corresponding state



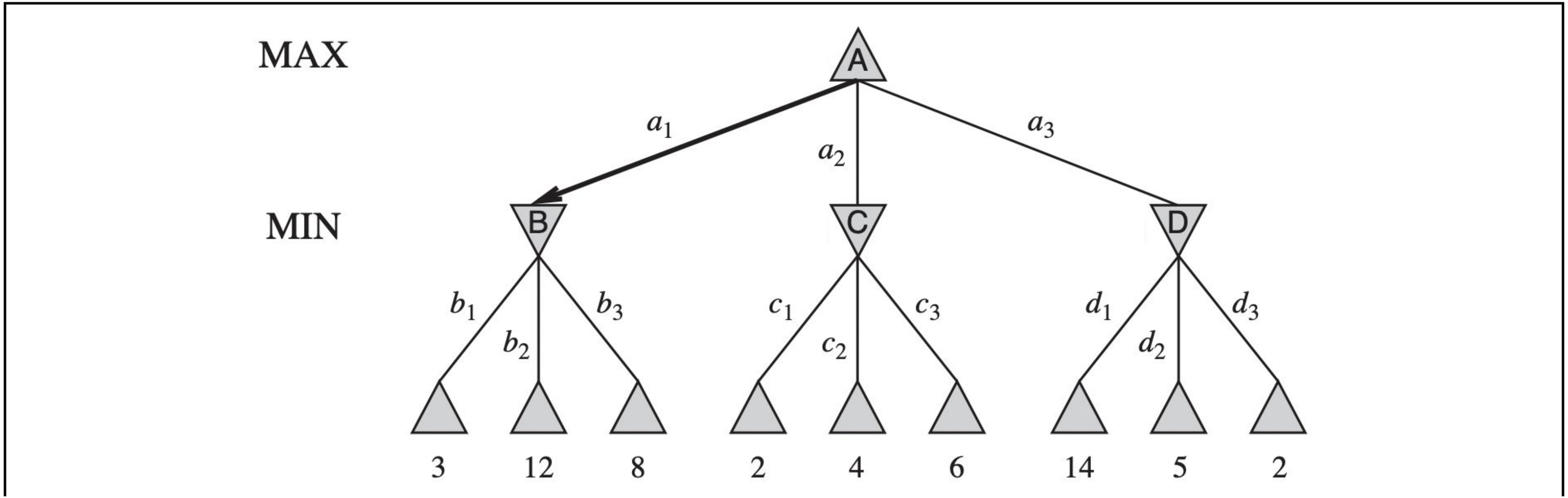
MiniMax Function

- The minimax value of a node is the utility (for MAX) of being in the corresponding state
- Assume that both players play optimally from there to the end of the game



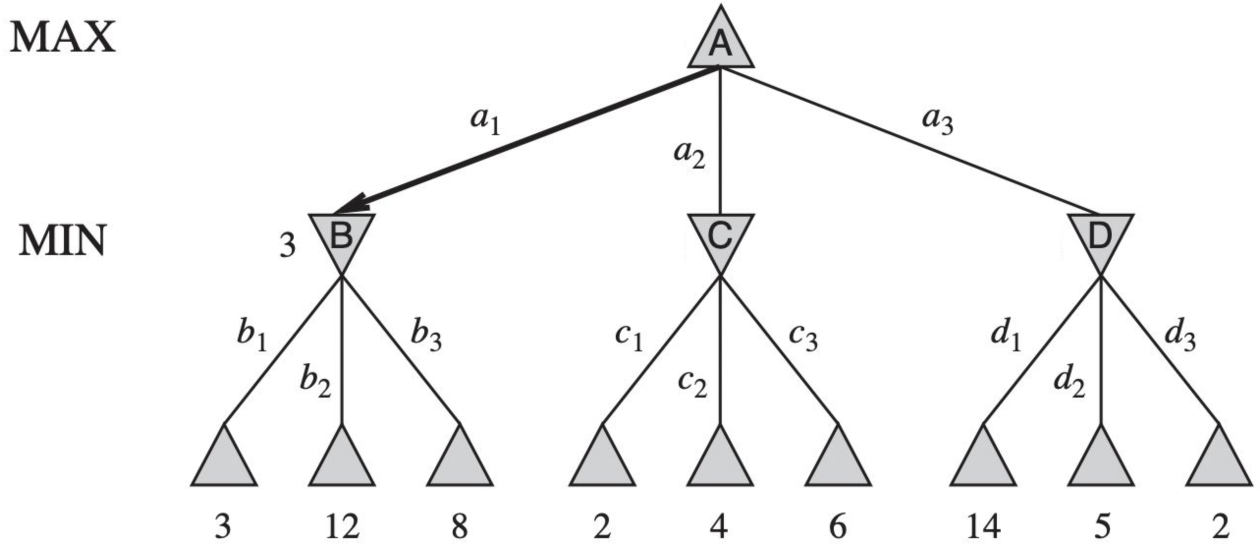
MiniMax Function

- The minimax value of a node is the utility (for MAX) of being in the corresponding state
- Assume that both players play optimally from there to the end of the game
- The minimax value of a terminal state is just its utility



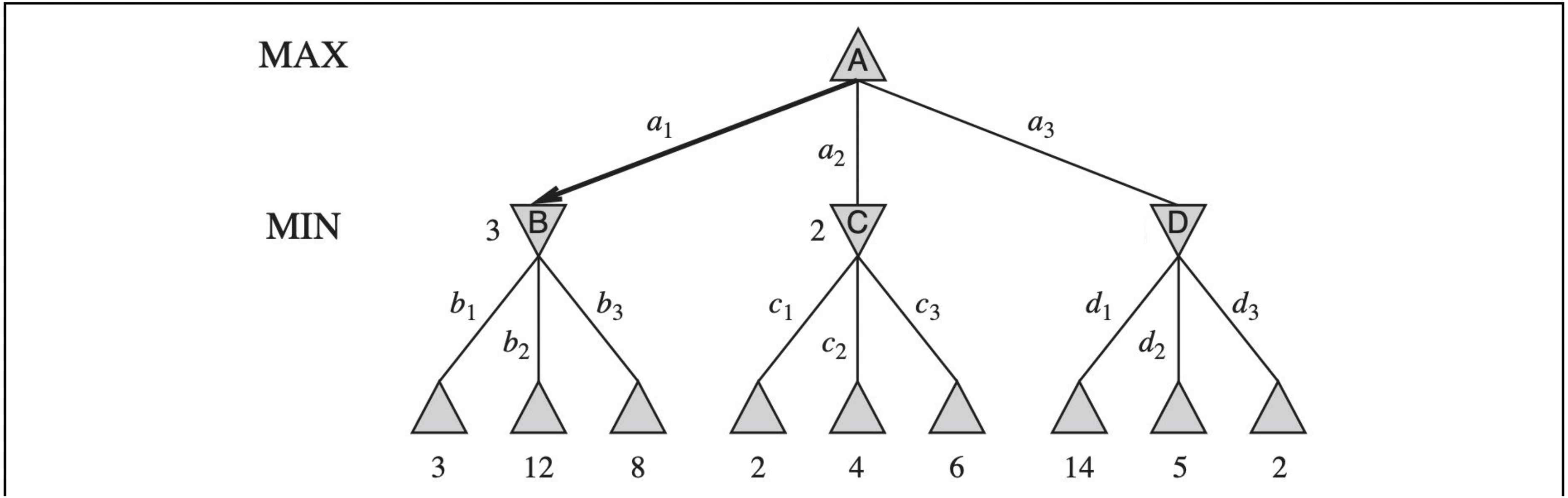
MiniMax Function

- The minimax value of a node is the utility (for MAX) of being in the corresponding state
- Assume that both players play optimally from there to the end of the game
- The minimax value of a terminal state is just its utility
- Given a choice, MAX prefers to move to a state of maximum value



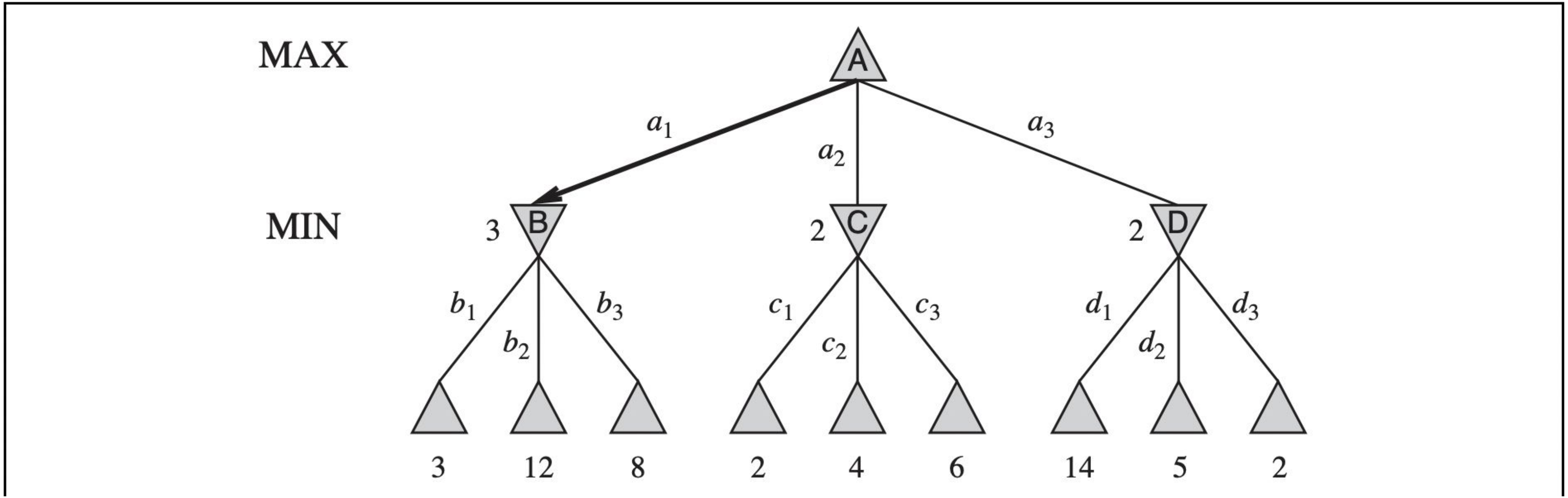
MiniMax Function

- The minimax value of a node is the utility (for MAX) of being in the corresponding state
- Assume that both players play optimally from there to the end of the game
- The minimax value of a terminal state is just its utility
- Given a choice, MAX prefers to move to a state of maximum value



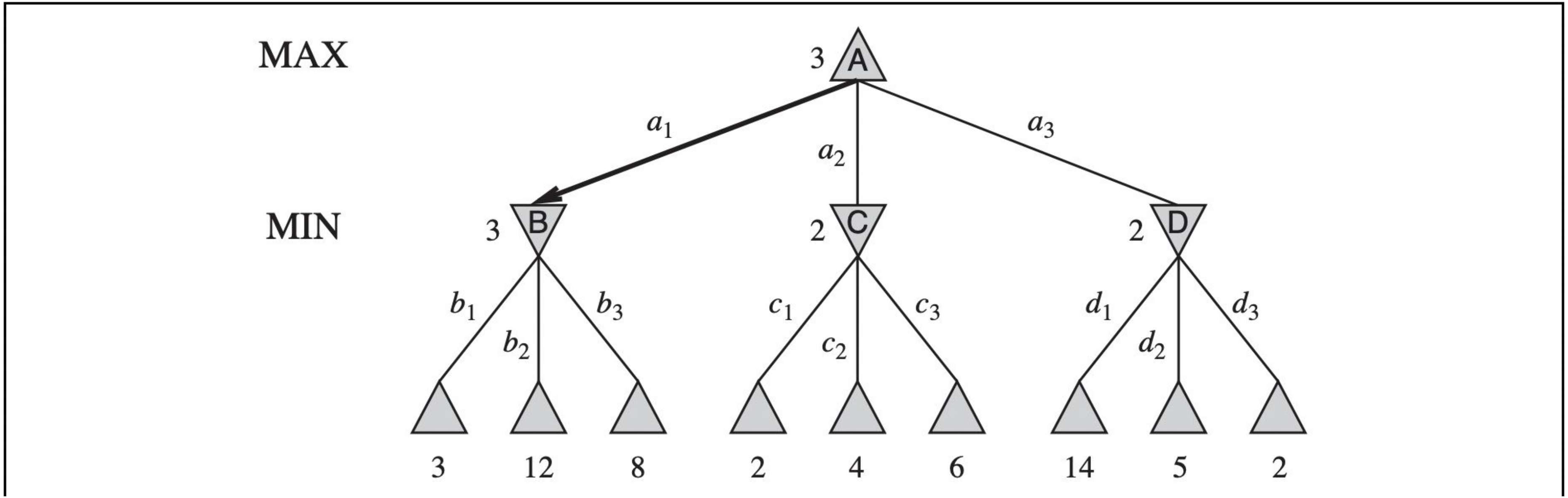
MiniMax Function

- The minimax value of a node is the utility (for MAX) of being in the corresponding state
- Assume that both players play optimally from there to the end of the game
- The minimax value of a terminal state is just its utility
- Given a choice, MAX prefers to move to a state of maximum value



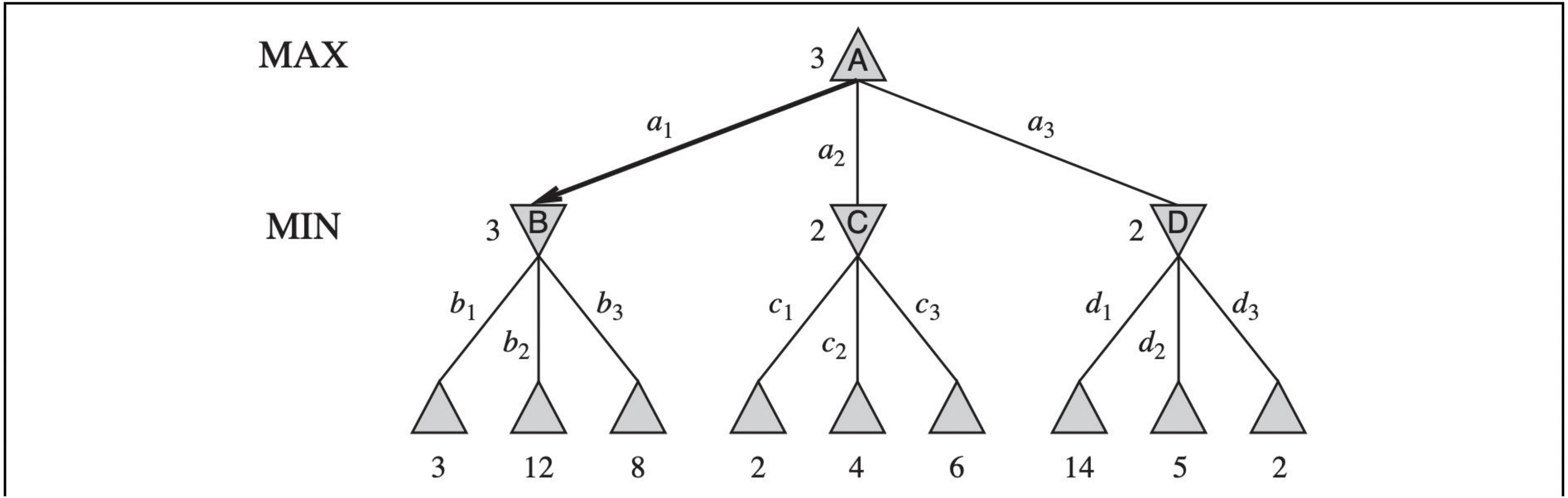
MiniMax Function

- The minimax value of a node is the utility (for MAX) of being in the corresponding state
- Assume that both players play optimally from there to the end of the game
- The minimax value of a terminal state is just its utility
- Given a choice, MAX prefers to move to a state of maximum value



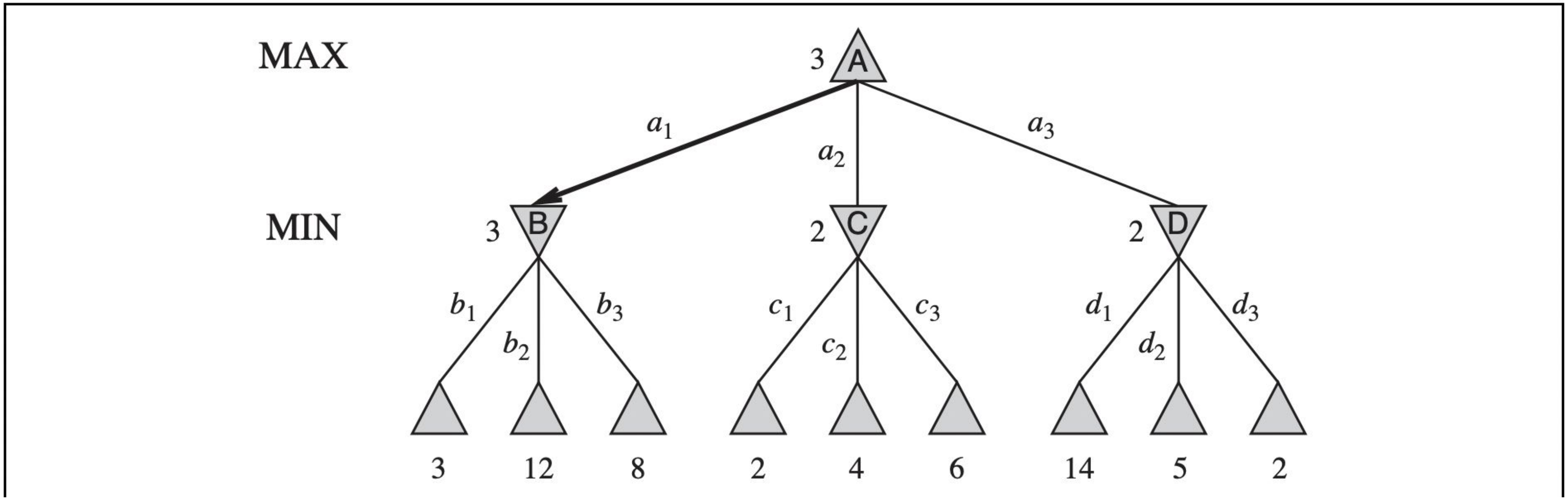
MiniMax Function

- The minimax value of a node is the utility (for MAX) of being in the corresponding state
- Assume that both players play optimally from there to the end of the game
- The minimax value of a terminal state is just its utility
- Given a choice, MAX prefers to move to a state of maximum value

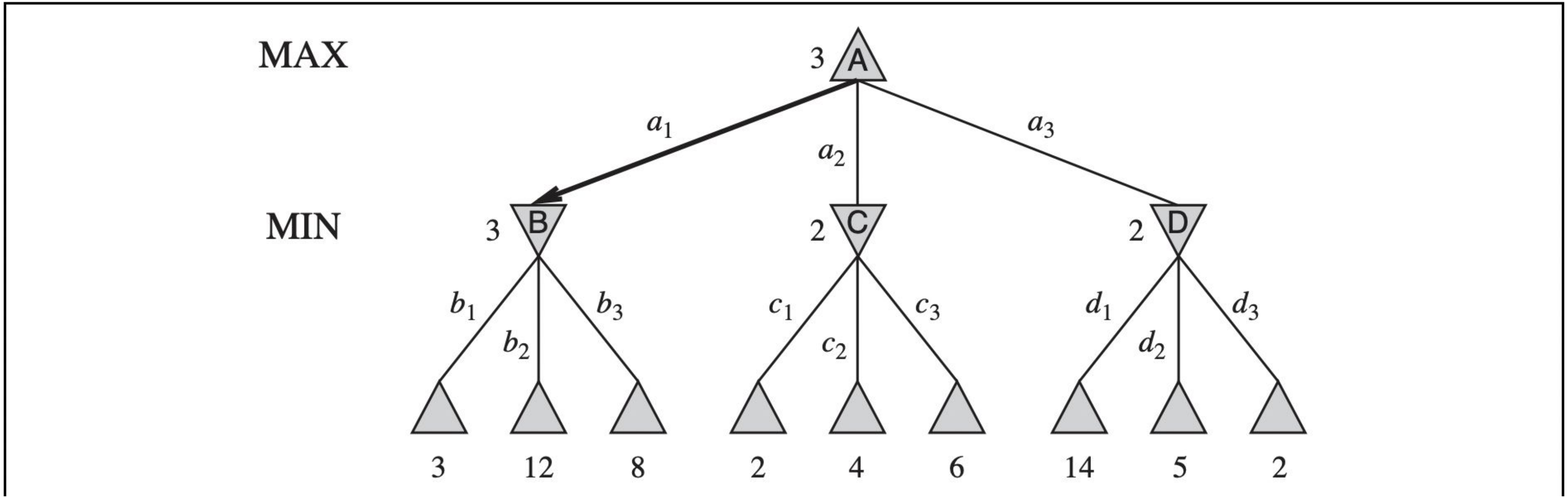


$\textsc{Minimax}(s) =$

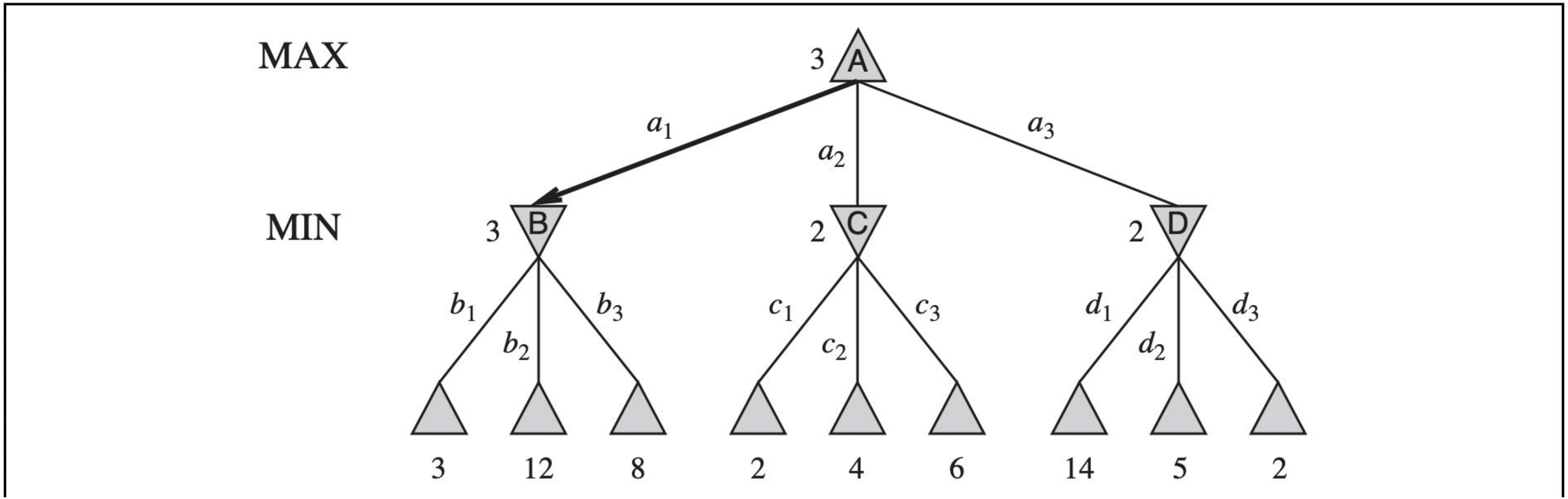
$\Bigg\{$



$$\textsc{Minimax}(s) =$$
$$\begin{cases} \textsc{Utility}(s) & \text{if } \textsc{Terminal-Test}(s) \\ & \end{cases}$$



$$\textsc{Minimax}(s) = \begin{cases} \textsc{Utility}(s) & \text{if } \textsc{Terminal-Test}(s) \\ \max_{a \in Actions(s)} \textsc{Minimax}(\textsc{Result}(s, a)) & \text{if } \textsc{Player}(s) = \textsc{max} \end{cases}$$



$$\textsc{Minimax}(s) = \begin{cases} \textsc{Utility}(s) & \text{if } \textsc{Terminal-Test}(s) \\ \max_{a \in Actions(s)} \textsc{Minimax}(\textsc{Result}(s,a)) & \text{if } \textsc{Player}(s) = \textsc{max} \\ \min_{a \in Actions(s)} \textsc{Minimax}(\textsc{Result}(s,a)) & \text{if } \textsc{Player}(s) = \textsc{min} \end{cases}$$

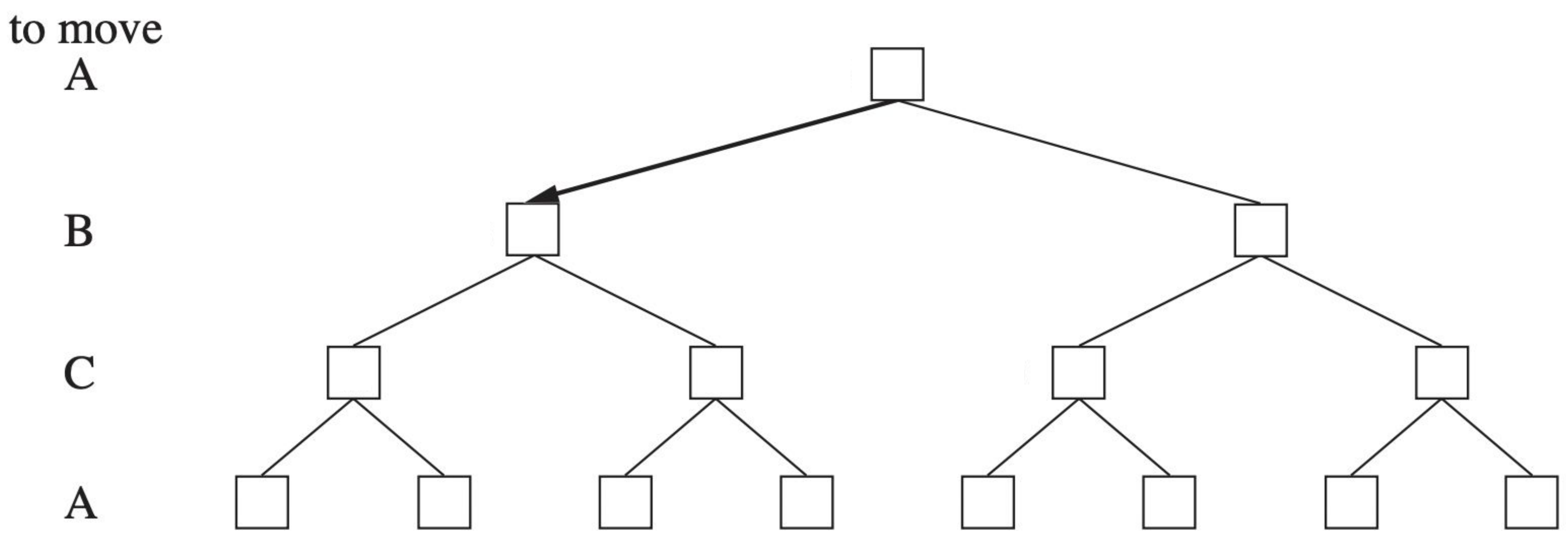


Figure 5.4 The first three plies of a game tree with three players (A, B, C). Each node is labeled with values from the viewpoint of each player. The best move is marked at the root.

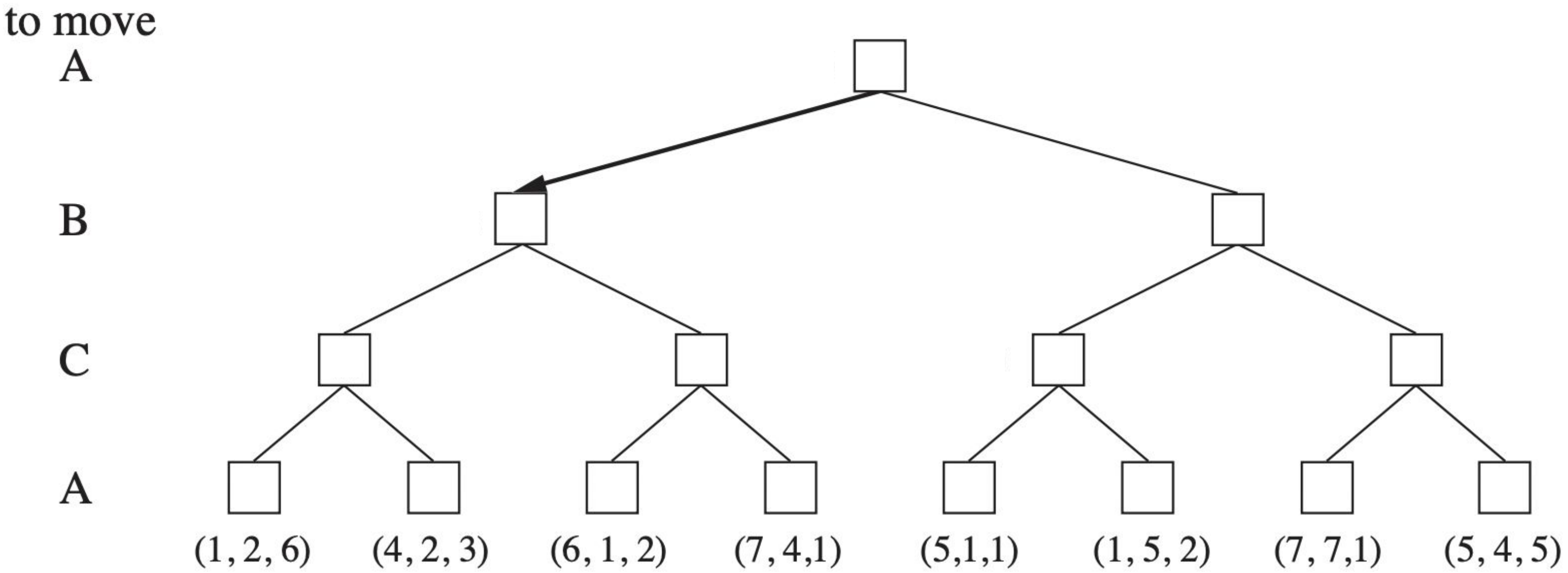


Figure 5.4 The first three plies of a game tree with three players (A, B, C). Each node is labeled with values from the viewpoint of each player. The best move is marked at the root.

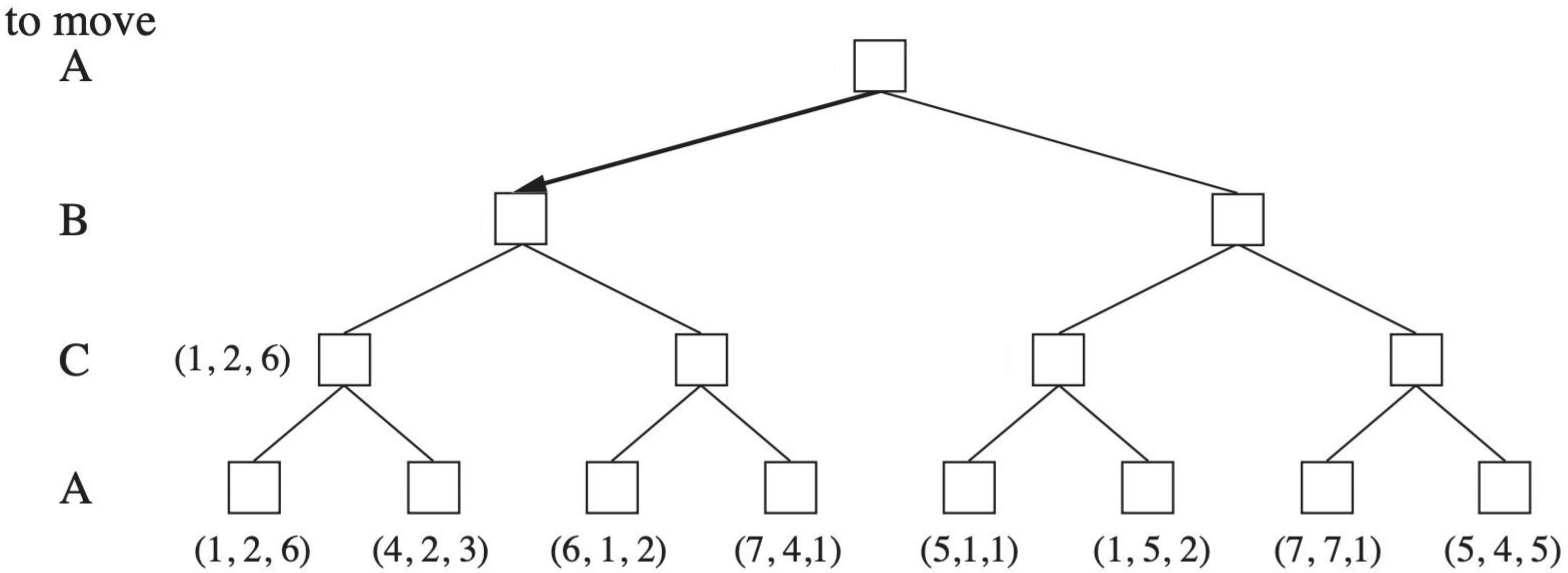


Figure 5.4 The first three plies of a game tree with three players (A, B, C). Each node is labeled with values from the viewpoint of each player. The best move is marked at the root.

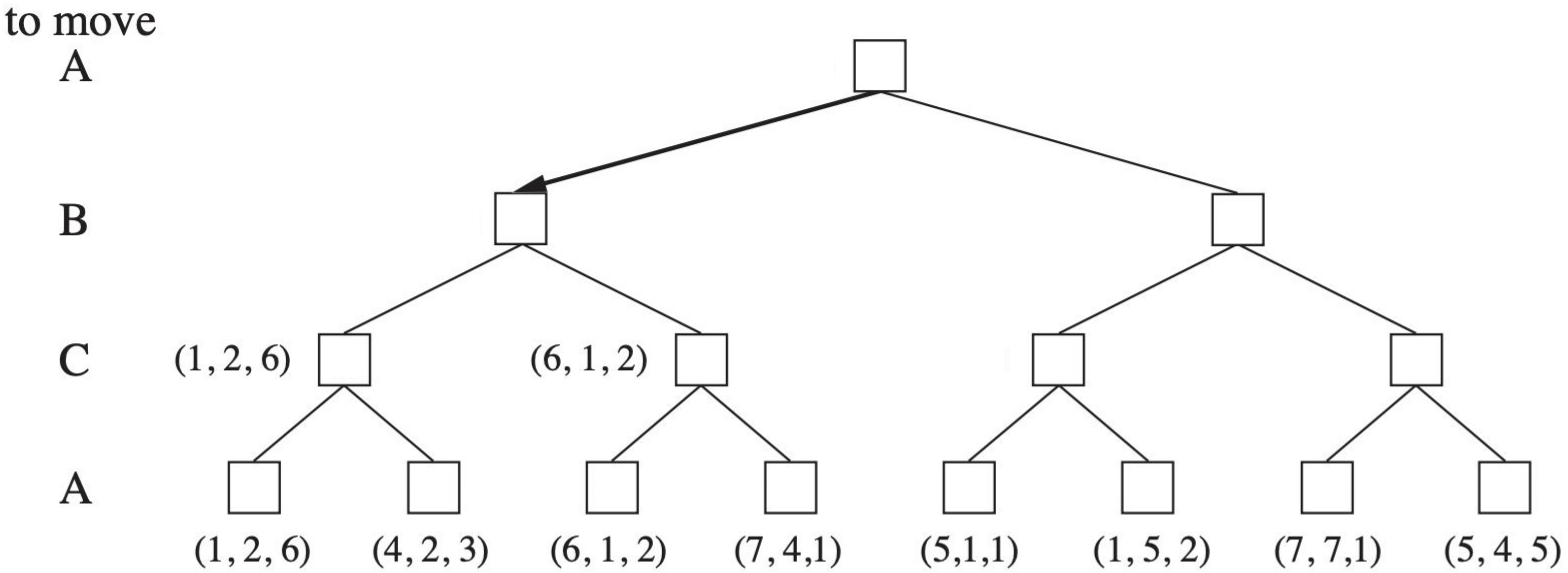


Figure 5.4 The first three plies of a game tree with three players (A, B, C). Each node is labeled with values from the viewpoint of each player. The best move is marked at the root.

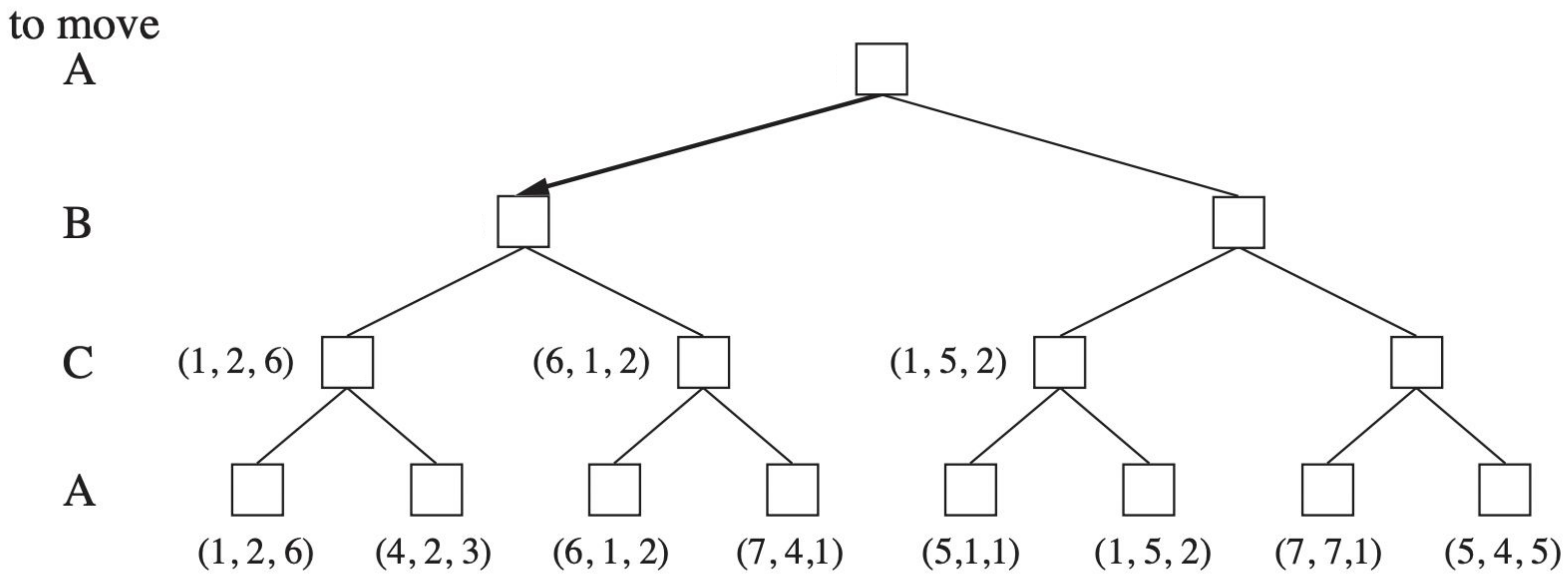


Figure 5.4 The first three plies of a game tree with three players (A, B, C). Each node is labeled with values from the viewpoint of each player. The best move is marked at the root.

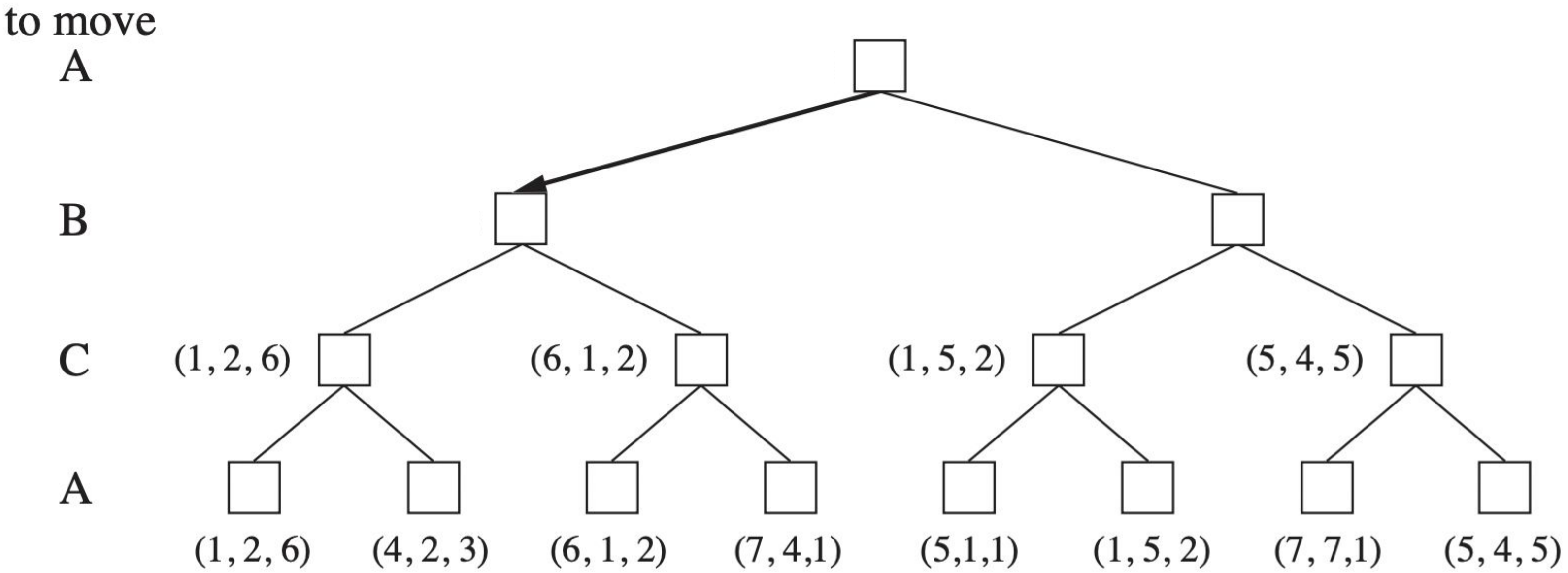


Figure 5.4 The first three plies of a game tree with three players (A, B, C). Each node is labeled with values from the viewpoint of each player. The best move is marked at the root.

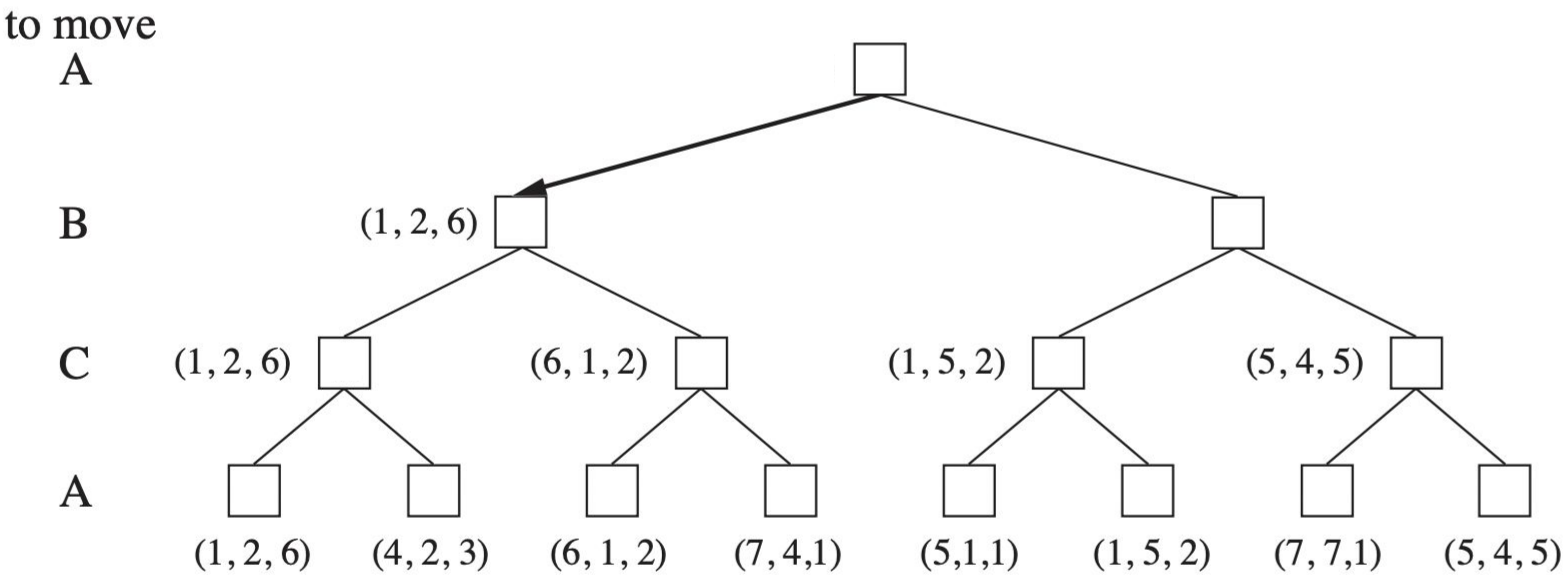


Figure 5.4 The first three plies of a game tree with three players (A, B, C). Each node is labeled with values from the viewpoint of each player. The best move is marked at the root.

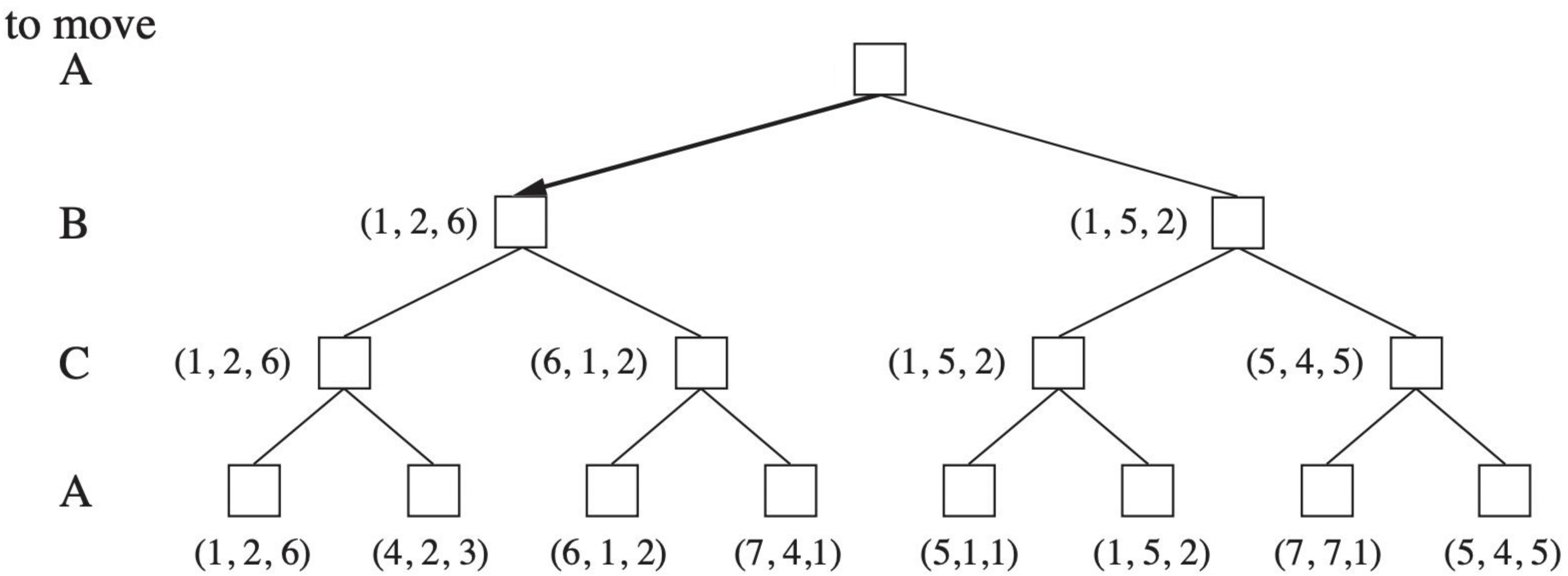


Figure 5.4 The first three plies of a game tree with three players (A, B, C). Each node is labeled with values from the viewpoint of each player. The best move is marked at the root.

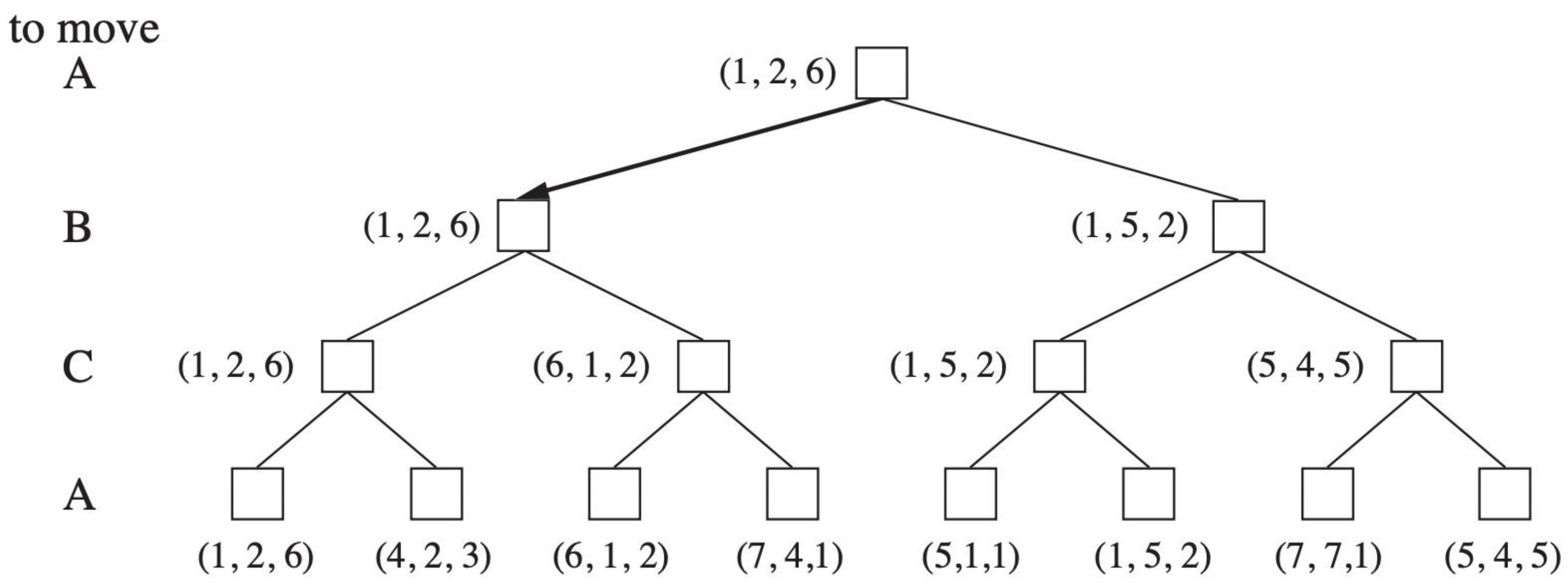


Figure 5.4 The first three plies of a game tree with three players (A, B, C). Each node is labeled with values from the viewpoint of each player. The best move is marked at the root.

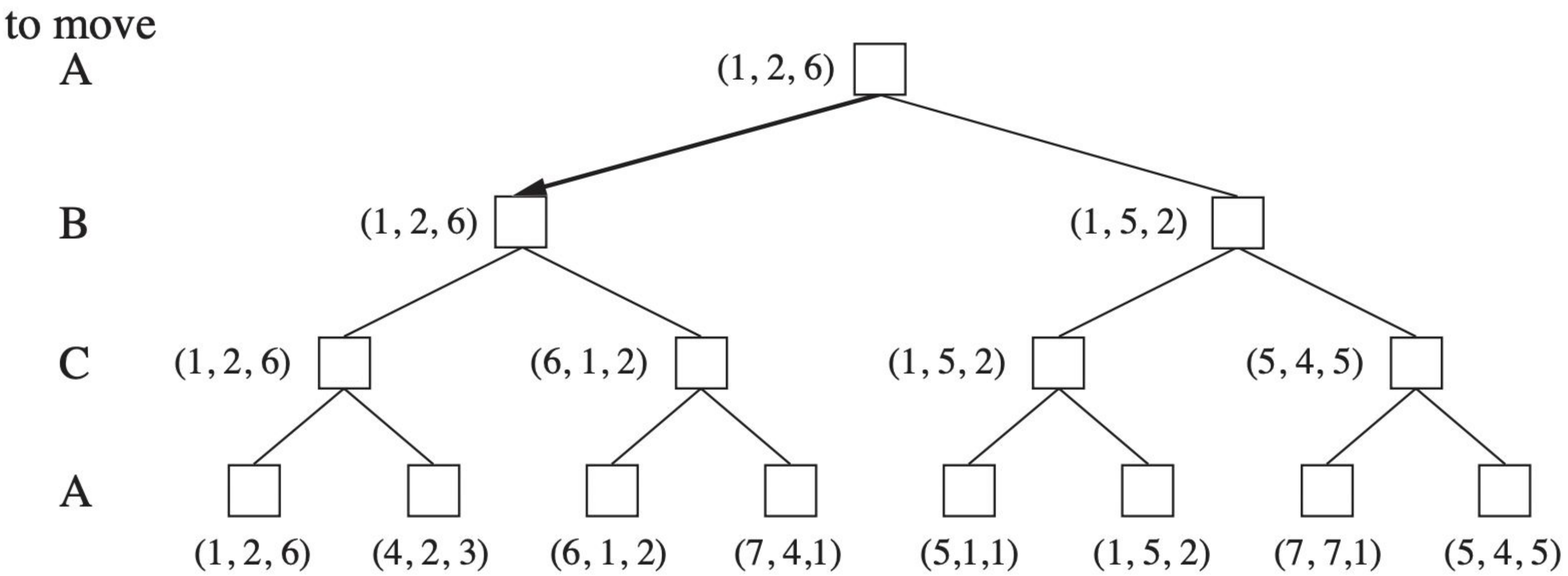


Figure 5.4 The first three plies of a game tree with three players (A, B, C). Each node is labeled with values from the viewpoint of each player. The best move is marked at the root.

- In a zero-sum three player game, do all the players have to always compete?

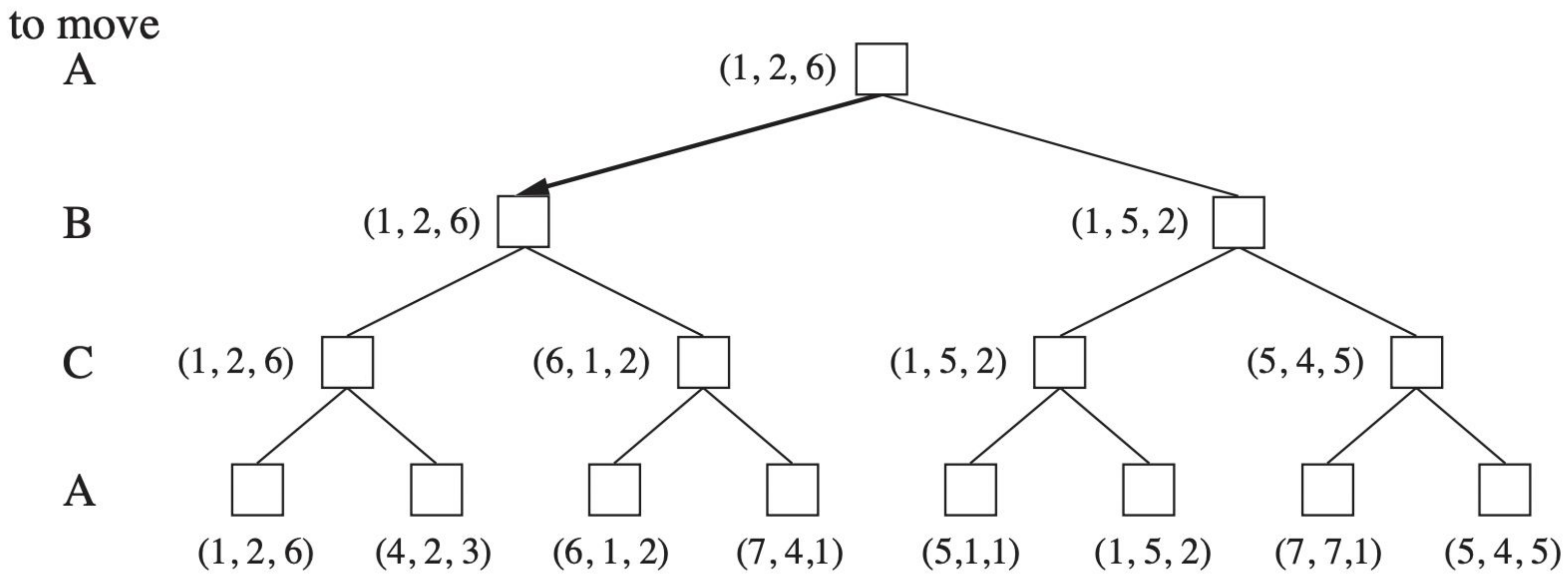


Figure 5.4 The first three plies of a game tree with three players (A, B, C). Each node is labeled with values from the viewpoint of each player. The best move is marked at the root.

- In a zero-sum three player game, do all the players have to always compete?
- If C is in a much stronger position, it makes sense for A and B to cooperate till they can pull down C

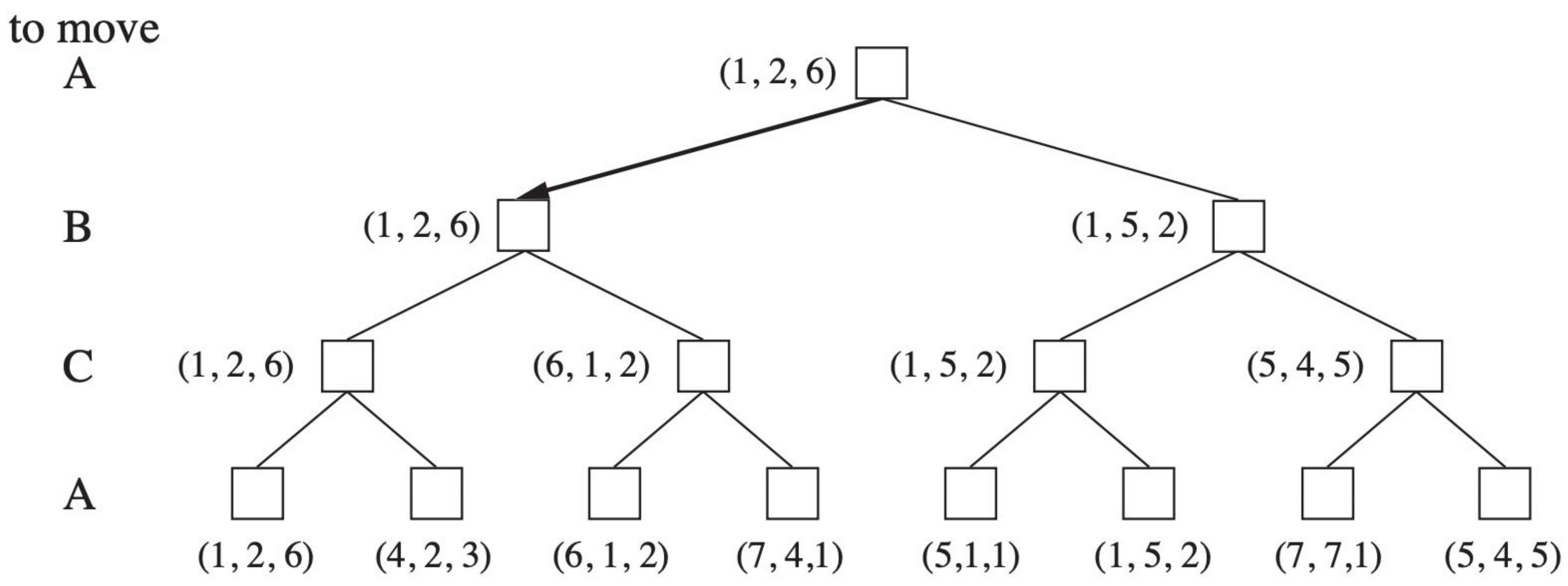
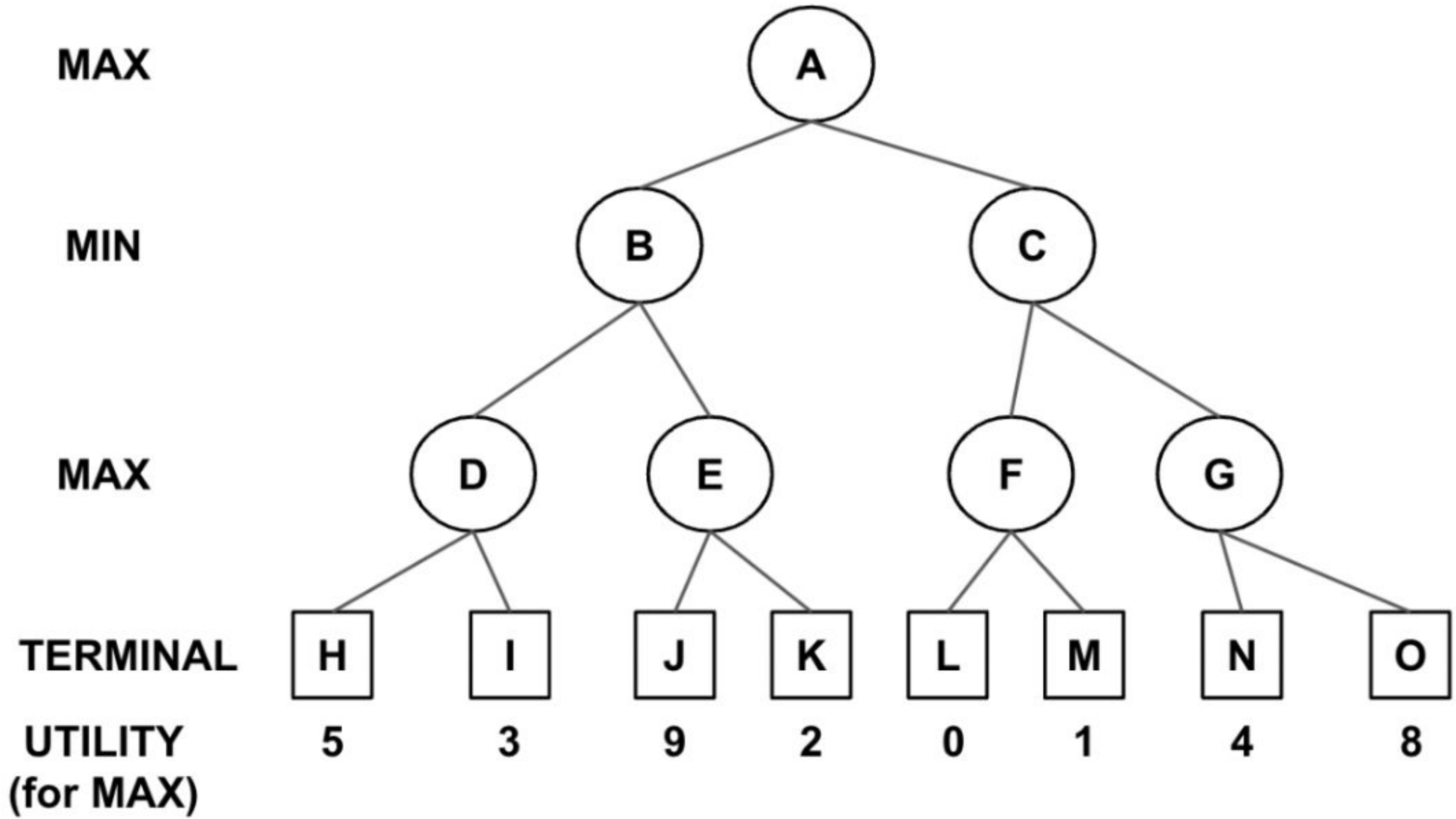
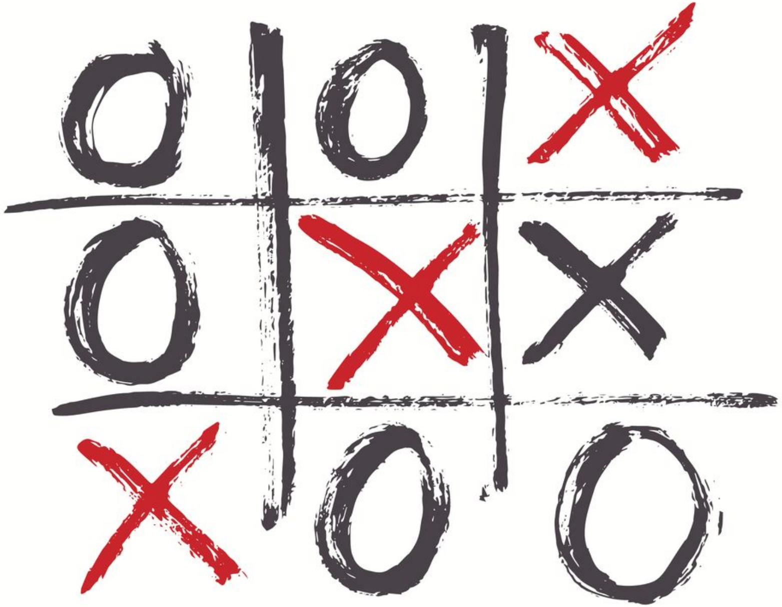


Figure 5.4 The first three plies of a game tree with three players (A, B, C). Each node is labeled with values from the viewpoint of each player. The best move is marked at the root.

- In a zero-sum three player game, do all the players have to always compete?
- If C is in a much stronger position, it makes sense for A and B to cooperate till they can pull down C [usually called politics, but its just game play in an organisation or even in families]

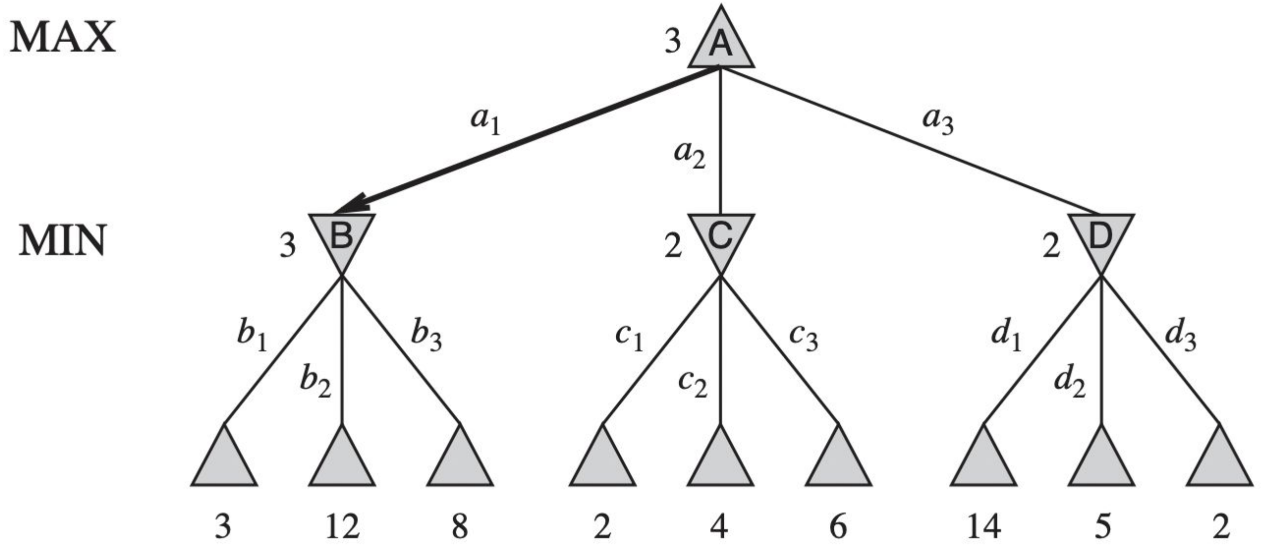
What is the MiniMax value at the root node (A) for this 2-player 3-ply game?





Write a python code
to generate the game tree for Tic-Tac-Toe,
and apply the MiniMax algorithm.

ALPHA-BETA PRUNING



MiniMax Function

- The minimax value of a node is the utility (for MAX) of being in the corresponding state
- Assume that both players play optimally from there to the end of the game
- The minimax value of a terminal state is just its utility
- Given a choice, MAX prefers to move to a state of maximum value

(a)

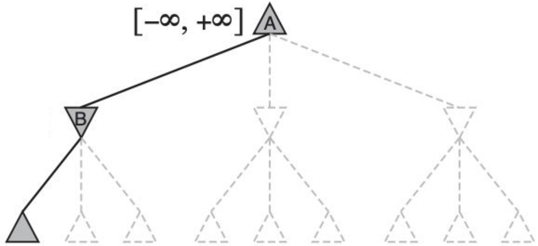


Figure 5.5 Stages in the calculation of the optimal decision for the game tree in Figure 5.2.

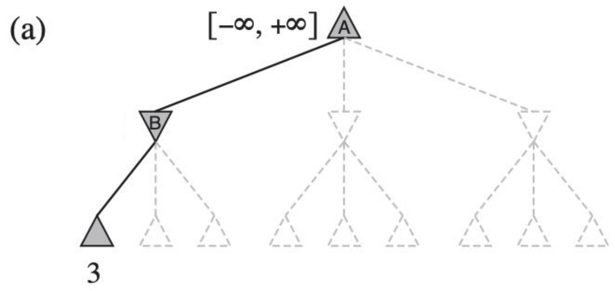


Figure 5.5 Stages in the calculation of the optimal decision for the game tree in Figure 5.2.

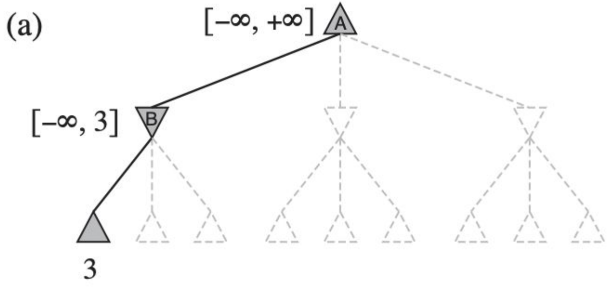


Figure 5.5 Stages in the calculation of the optimal decision for the game tree in Figure 5.2.

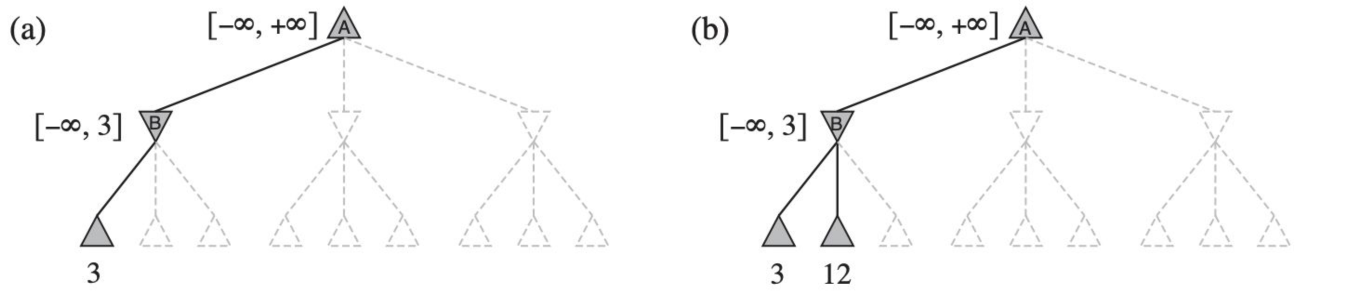


Figure 5.5 Stages in the calculation of the optimal decision for the game tree in Figure 5.2.

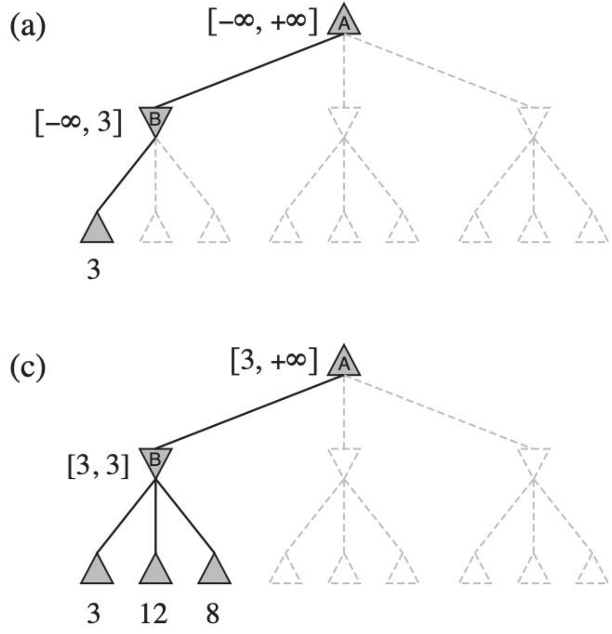


Figure 5.5 Stages in the calculation of the optimal decision for the game tree in Figure 5.2.

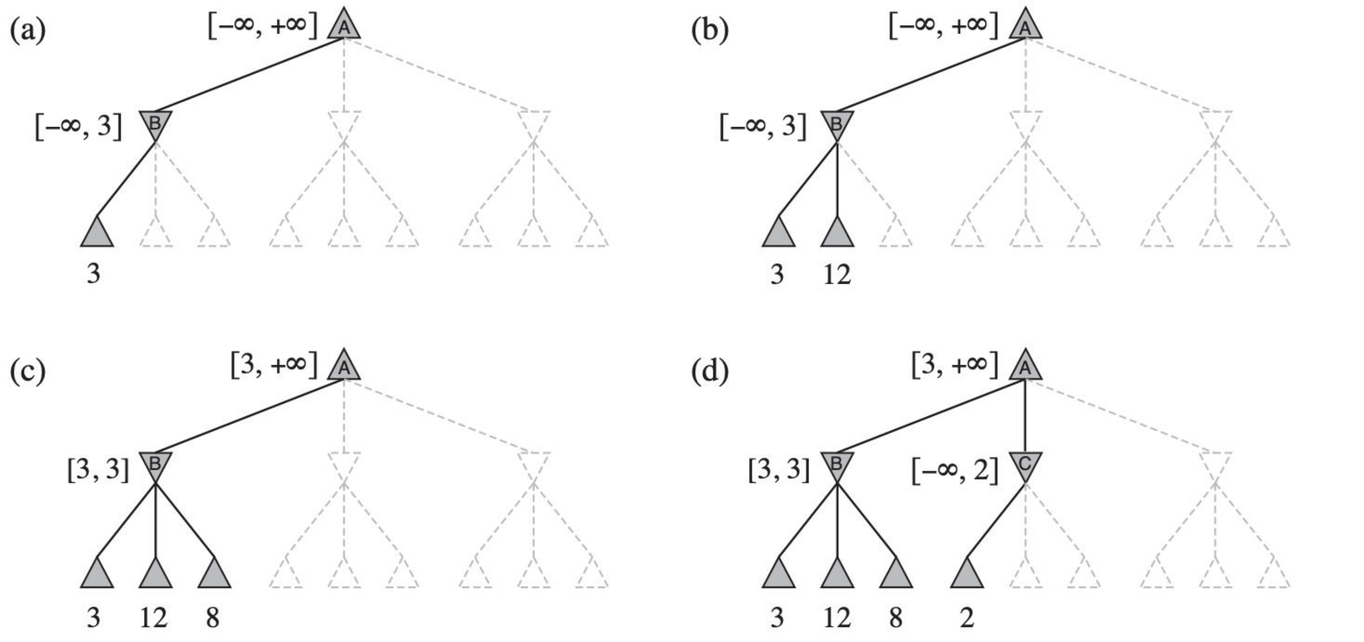


Figure 5.5 Stages in the calculation of the optimal decision for the game tree in Figure 5.2.

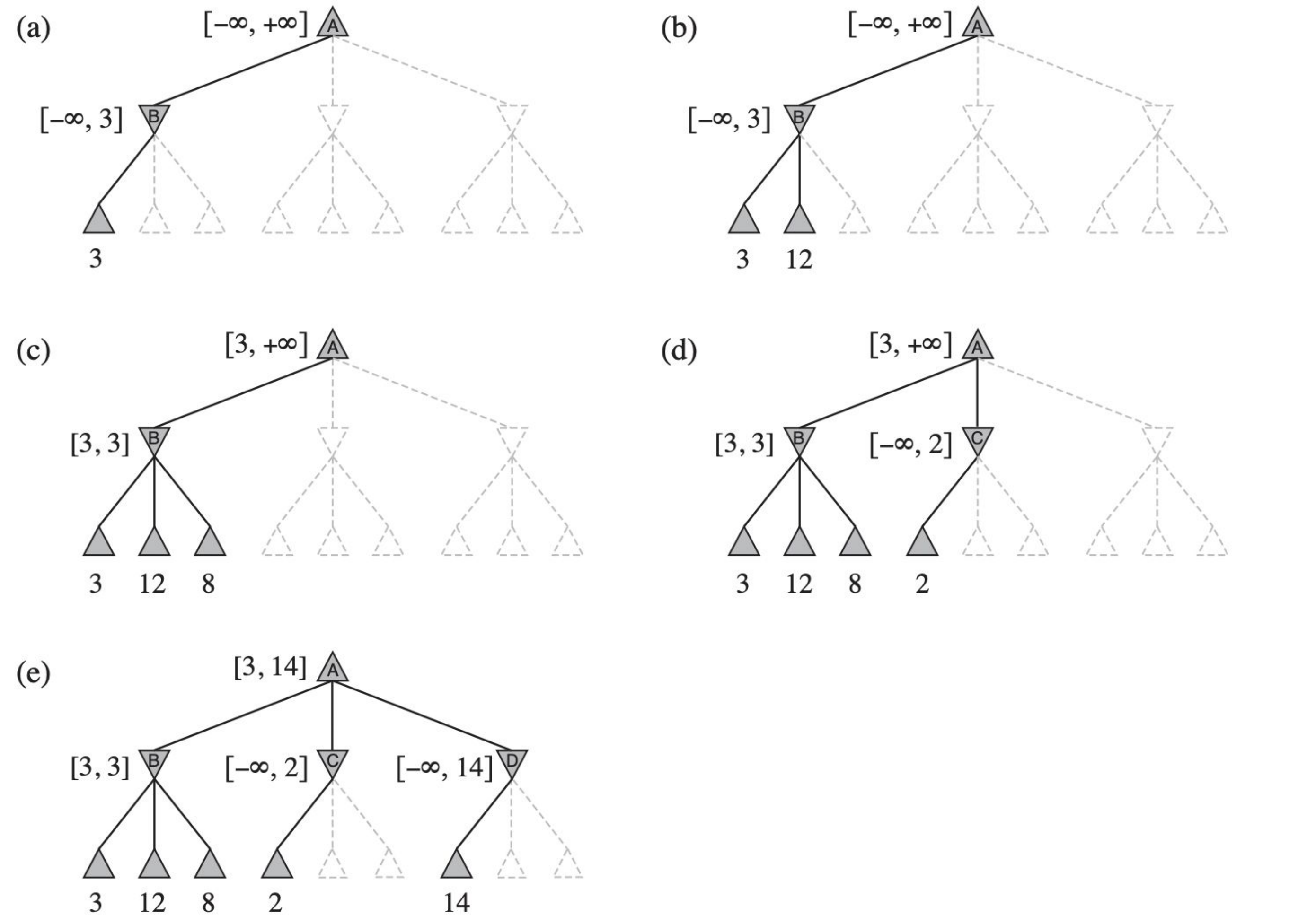


Figure 5.5 Stages in the calculation of the optimal decision for the game tree in Figure 5.2.

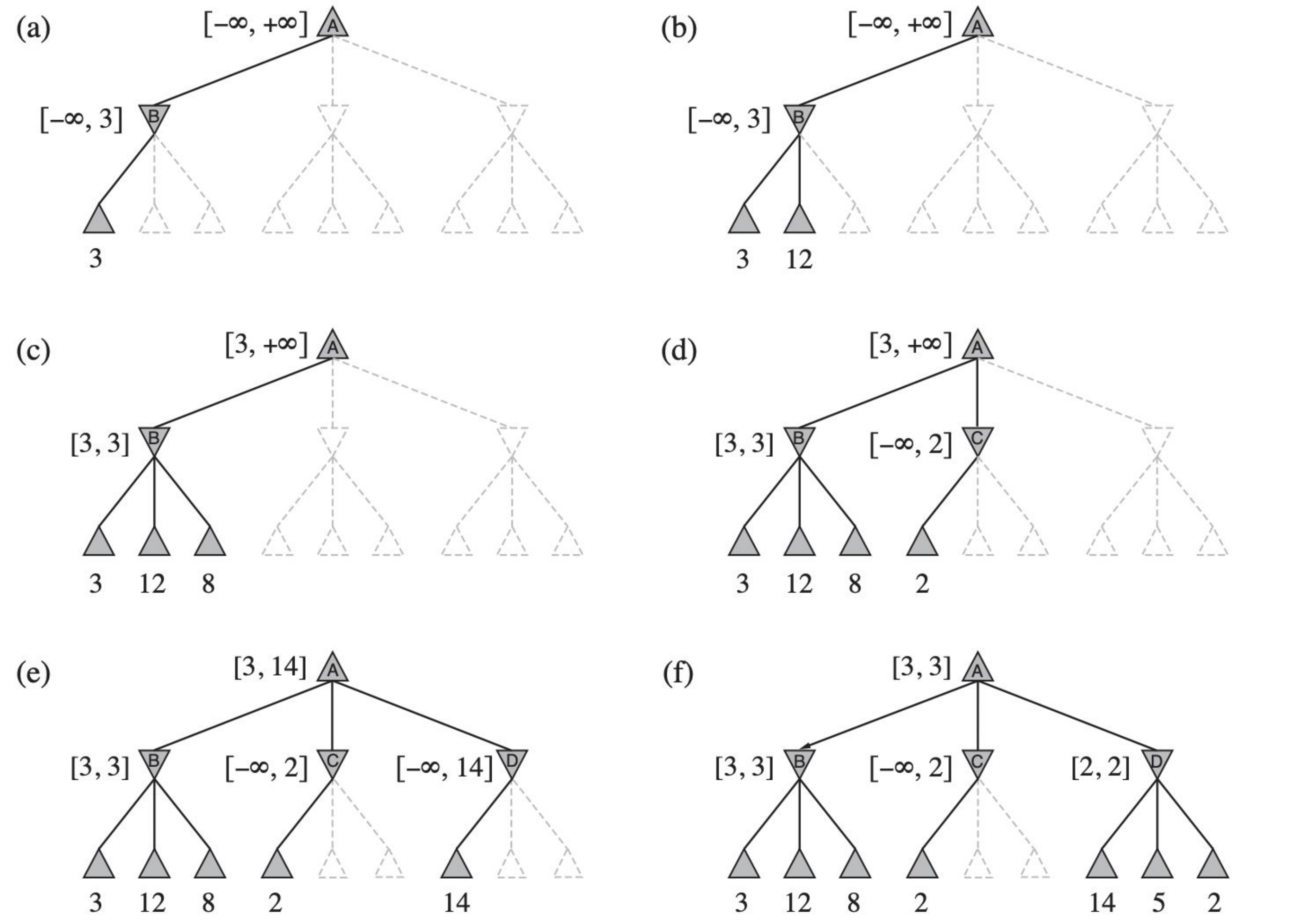
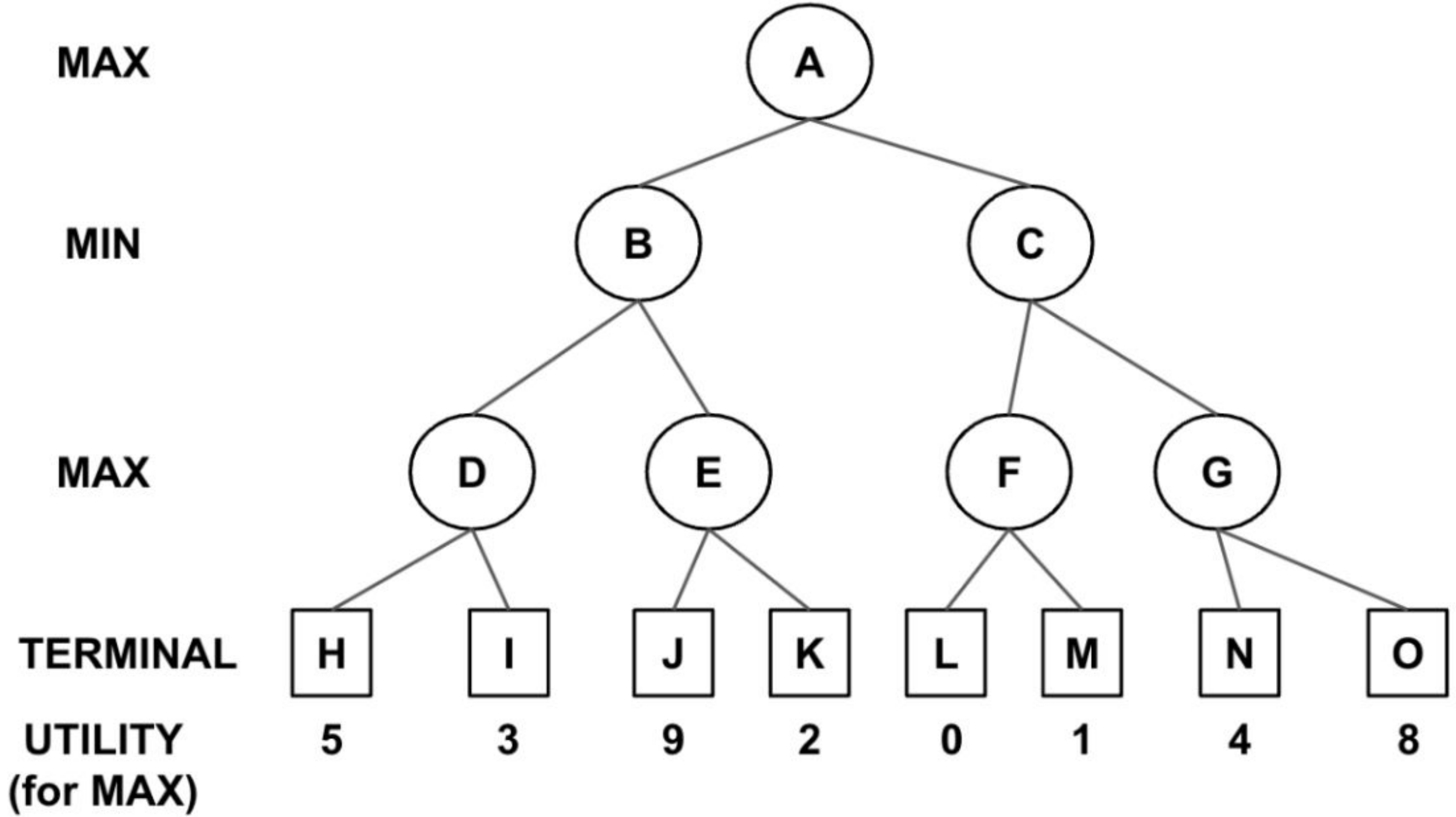


Figure 5.5 Stages in the calculation of the optimal decision for the game tree in Figure 5.2.

If we use alpha-beta pruning, which nodes will not be visited by the algorithm?



One of the two cabinets of Deep Blue in its exhibit at the Computer History Museum, California

Active	1995 (prototype) 1996 (release) 1997 (upgrade)

- IBM Deep Blue was the first machine to beat a reigning world chess champion in a six-game match

One of the two cabinets of Deep Blue in its exhibit at the Computer History Museum, California

Active	1995 (prototype) 1996 (release) 1997 (upgrade)

- IBM Deep Blue was the first machine to beat a reigning world chess champion in a six-game match

- Factors that contributed to this success:
    - single-chip chess search engine
    - massively parallel system
    - strong emphasis on search extensions
    - complex evaluation function
        - Alpha-Beta Pruning
        - Iterative Deepening
    - effective use of a Grandmaster game database

One of the two cabinets of Deep Blue in its exhibit at the Computer History Museum, California

Active	1995 (prototype) 1996 (release) 1997 (upgrade)

- IBM Deep Blue was the first machine to beat a reigning world chess champion in a six-game match

- Factors that contributed to this success:
    - single-chip chess search engine
    - massively parallel system
    - strong emphasis on search extensions
    - complex evaluation function
        - Alpha-Beta Pruning
        - Iterative Deepening
    - effective use of a Grandmaster game database

One of the two cabinets of Deep Blue in its exhibit at the Computer History Museum, California

Active	1995 (prototype) 1996 (release) 1997 (upgrade)

- IBM Deep Blue was the first machine to beat a reigning world chess champion in a six-game match

- Factors that contributed to this success:
  - single-chip chess search engine
  - massively parallel system
  - strong emphasis on search extensions
  - complex evaluation function
    - Alpha-Beta Pruning
    - Iterative Deepening
  - effective use of a Grandmaster game database

- Kasparov demanded a rematch, but IBM had dismantled Deep Blue after its victory and refused the rematch

One of the two cabinets of Deep Blue in its exhibit at the Computer History Museum, California

Active	1995 (prototype) 1996 (release) 1997 (upgrade)

- IBM Deep Blue was the first machine to beat a reigning world chess champion in a six-game match

- Factors that contributed to this success:
    - single-chip chess search engine
    - massively parallel system
    - strong emphasis on search extensions
    - complex evaluation function
        - Alpha-Beta Pruning
        - Iterative Deepening
    - effective use of a Grandmaster game database

- Kasparov demanded a rematch, but IBM had dismantled Deep Blue after its victory and refused the rematch

- Inspired the development of IBM Watson for playing Jeopardy

One of the two cabinets of Deep Blue in its exhibit at the Computer History Museum, California

Active	1995 (prototype) 1996 (release) 1997 (upgrade)

- IBM Deep Blue was the first machine to beat a reigning world chess champion in a six-game match

- Factors that contributed to this success:
  - single-chip chess search engine
  - massively parallel system
  - strong emphasis on search extensions
  - complex evaluation function
    - Alpha-Beta Pruning
    - Iterative Deepening
  - effective use of a Grandmaster game database

- Kasparov demanded a rematch, but IBM had dismantled Deep Blue after its victory and refused the rematch

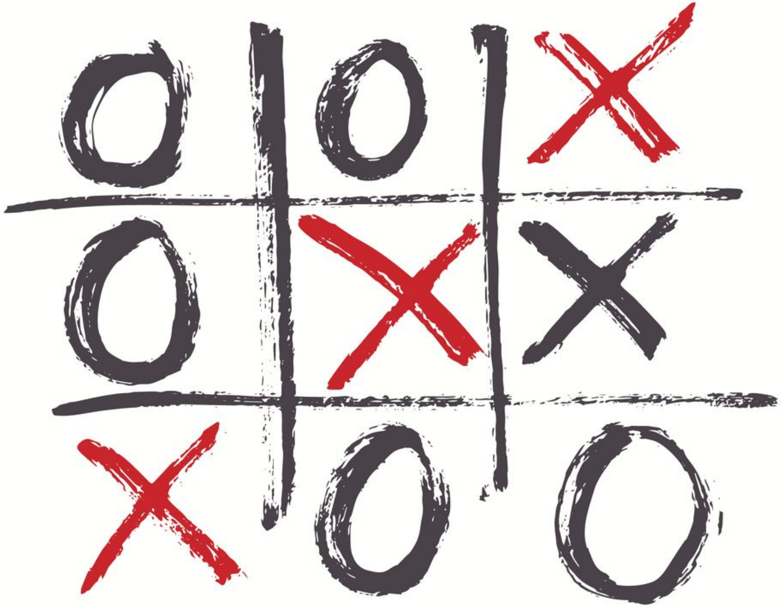
- Inspired the development of IBM Watson for playing Jeopardy

- Alpha-beta pruning worked great for Chess, Checkers and Othello

One of the two cabinets of Deep Blue in its exhibit at the Computer History Museum, California

Active	1995 (prototype) 1996 (release) 1997 (upgrade)

- IBM Deep Blue was the first machine to beat a reigning world chess champion in a six-game match
- Factors that contributed to this success:
  - single-chip chess search engine
  - massively parallel system
  - strong emphasis on search extensions
  - complex evaluation function
    - Alpha-Beta Pruning
    - Iterative Deepening
  - effective use of a Grandmaster game database
- Kasparov demanded a rematch, but IBM had dismantled Deep Blue after its victory and refused the rematch
- Inspired the development of IBM Watson for playing Jeopardy
- Alpha-beta pruning worked great for Chess, Checkers and Othello
- AlphaGo uses Monte Carlo Tree Search (MCTS)



Write a python code
to generate the game tree for Tic-Tac-Toe,
and apply the Alpha-Beta pruning algorithm.