Artificial Intelligence

Adversarial Search

Motivation

- Kasparov vs Deep Blue
- Deep Blue wins by 3 wins, 1 loss and 2 draws in 1997

- AlphaGo by Google Deep Mind

- **Adversarial Search**: To find the optimal move or strategy for a player in a two-player game where each player is trying to maximize their chances of winning.
- In adversarial search, the algorithm assumes that the opponent is also playing optimally, and therefore, it tries to find the best possible move that will maximize the player's chances of winning, given the opponent's best possible move.

Game Playing

- Game Playing is a search problem defined by:
  - Initial State: board configuration of chess
  - Successor Function:
    - Player(s): two players A & B;
    - Actions: Legal moves in a state
    - Results: It defines the result of move
  - Terminal-Test: End of Game?

  Terminal test, which is true when the game is over and false otherwise. States where the game has ended are called **terminal states**.
  - Utility: Objective Function: it defines final numeric value for a game that ends in terminal state. E.g. win (+1), lose (-1) and draw (0) in chess

- Initial State, Actions and Results function define the game tree for a game.
  - Nodes are state
  - Edges are moves

Perfect Information (Zero-Sum) Game

- AI Games
  - Perfect Information
    - Deterministic
    - Fully Observable
  - Zero Sum Game
    - Utility Functions of each player at the end of the game are EQUAL and OPPOSITE
      - e.g, WINNER (1) , LOSER (-1) or multi-valued utility values
  - Giving rise to adversary … agents need to maximize their utility values

Zero Sum: any gain by one player is exactly balanced by a loss by the other player or players.
In other words, the total payoff is constant and any increase in one player's payoff is necessarily accompanied by a decrease in the other player's payoff.

Games vs. search problems

- "Unpredictable" opponent
  - specifying a move for every possible opponent reply
- Time limits
  - unlikely to find goal, must approximate

Game tree (2-player, deterministic, turns)

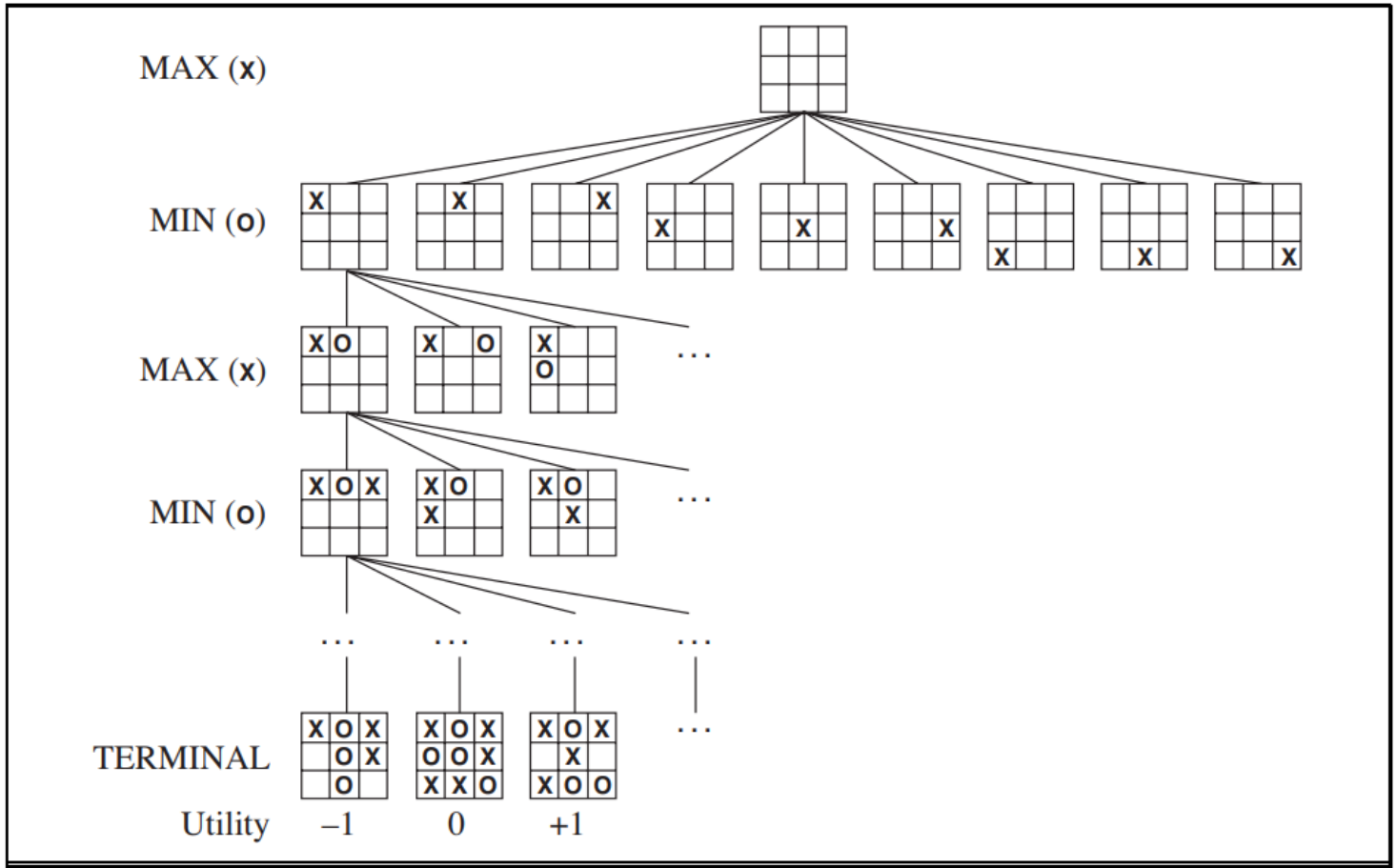


Figure 5.1 A (partial) game tree for the game of tic-tac-toe. The top node is the initial state, and MAX moves first, placing an X in an empty square. We show part of the tree, giving alternating moves by MIN (O) and MAX (X), until we eventually reach terminal states, which can be assigned utilities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

Optimal Decisions in Games

- Search problem formulation for MINMAX
  - Initial State / Successor Function / Terminal State / Utility Fn
- Explanation
  - MAX starts first
  - Play alternates b/w MAX (places X) & MIN (places O)
  - Terminal State (WIN/LOSS/DRAW)
  - Number each leaf node w.r.t MAX
    - High values are GOOD for MAX, BAD for MIN
    - It is MAX's job to use search-tree to determine "best move"
- Normal Search solution vs. Game's Solution
    - Normal sequence of moves leading to goal vs. MIN has a say
    - Requires contingent OPTIMAL strategy (in response to MIN's move)

Minimax

- Perfect play for deterministic games
- Idea: choose move to position with highest minimax value=best achievable payoff against best play
- Use DFS for searching
- Why not BFS?

Minimax

- Minimax Value=
  - Utility (n) if n is a terminal state
  - Max(n) set of successors if n is a MAX node
  - Min(n) set of successors if n is a MIN node

MiniMax “Search”

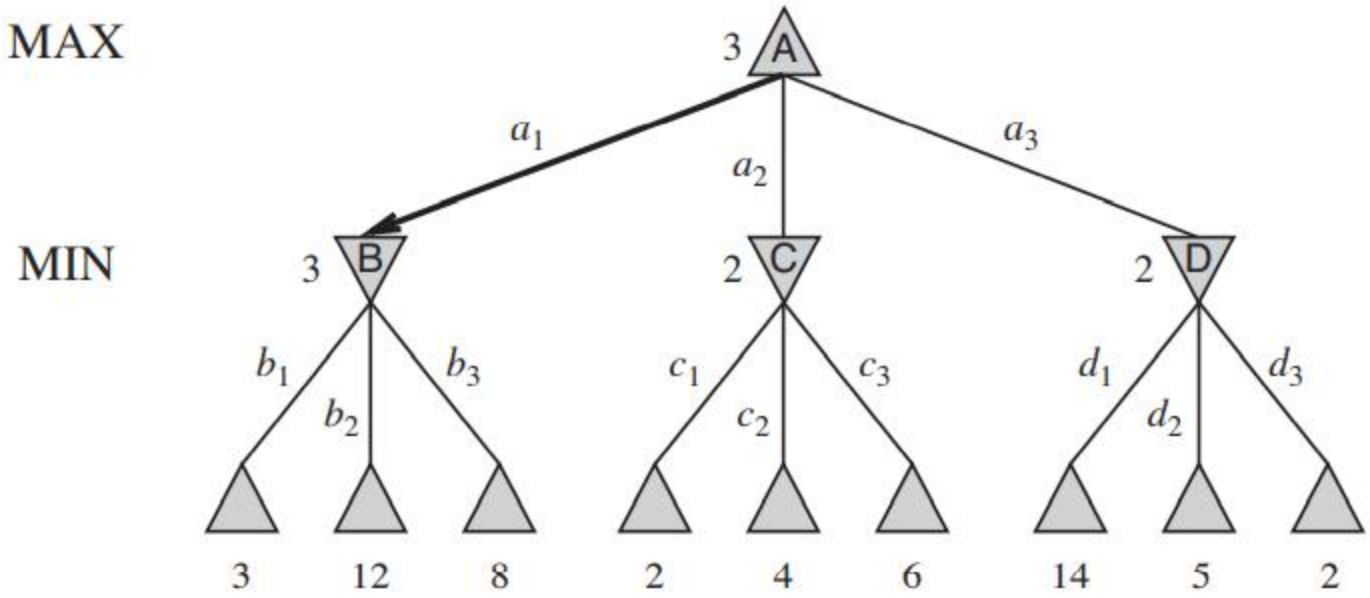


Figure 5.2 A two-ply game tree. The △ nodes are “MAX nodes,” in which it is MAX’s turn to move, and the ▽ nodes are “MIN nodes.” The terminal nodes show the utility values for MAX; the other nodes are labeled with their minimax values. MAX’s best move at the root is a_1, because it leads to the state with the highest minimax value, and MIN’s best reply is b_1, because it leads to the state with the lowest minimax value.

Minimax algorithm

```
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
```

Figure 5.3 An algorithm for calculating minimax decisions. It returns the action corresponding to the best possible move, that is, the move that leads to the outcome with the best utility, under the assumption that the opponent plays to minimize utility. The functions MAX-VALUE and MIN-VALUE go through the whole game tree, all the way to the leaves, to determine the backed-up value of a state. The notation $\text{argmax}_{a \in S} f(a)$ computes the element a of set S that has the maximum value of $f(a)$.

Properties of minimax

- Complete? Yes (if tree is finite)
- Optimal? Yes (against an optimal opponent)
- Time complexity? $O(b^d)$
- Space complexity? O(bd) (depth-first exploration)
- For chess, b ≈ 35, m ≈100 for "reasonable" games
  → exact solution completely infeasible

MiniMax (Multiplayer)

Multiplayer games usually involve alliances, whether formal or informal, among the players. Alliances are made and broken as the game proceeds

suppose A and B are in weak positions and C is in a stronger position. Then it is often optimal for both A and B to attack C rather than each other, lest C destroy each of them individually

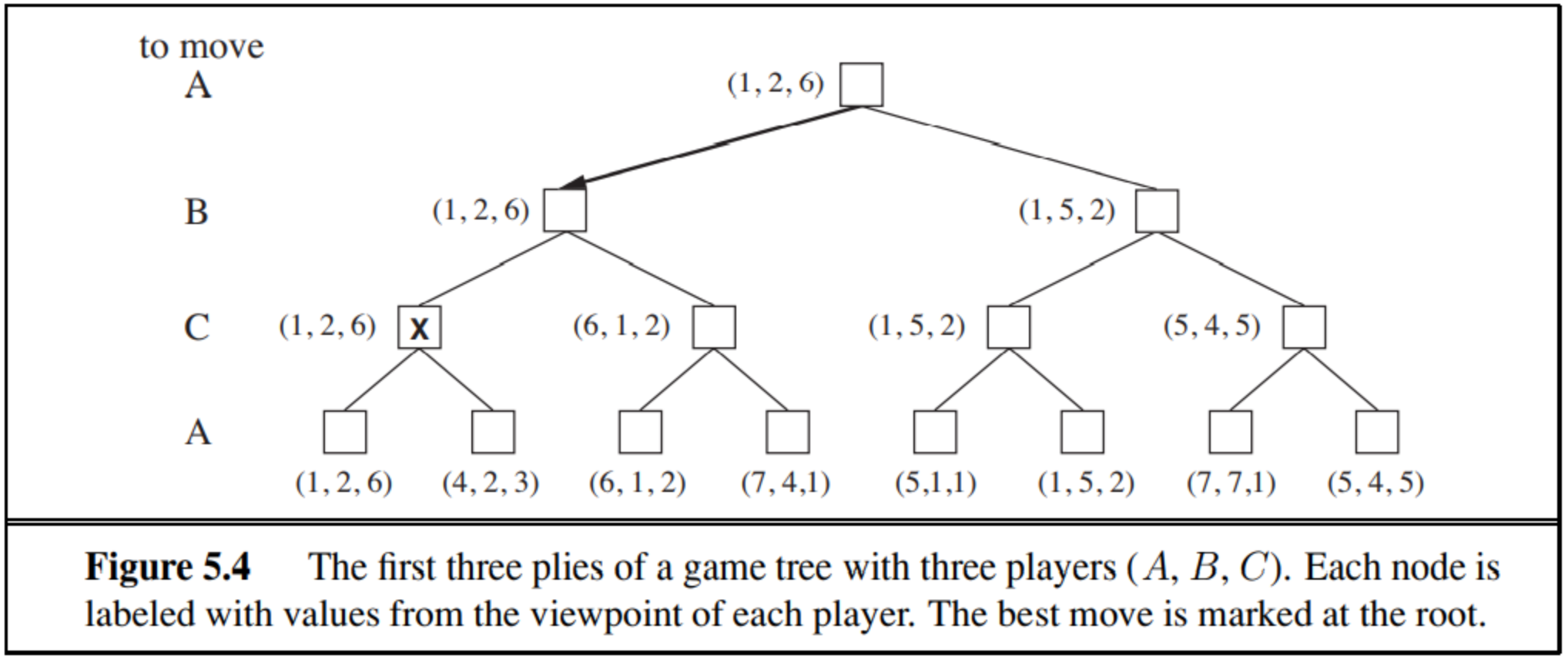
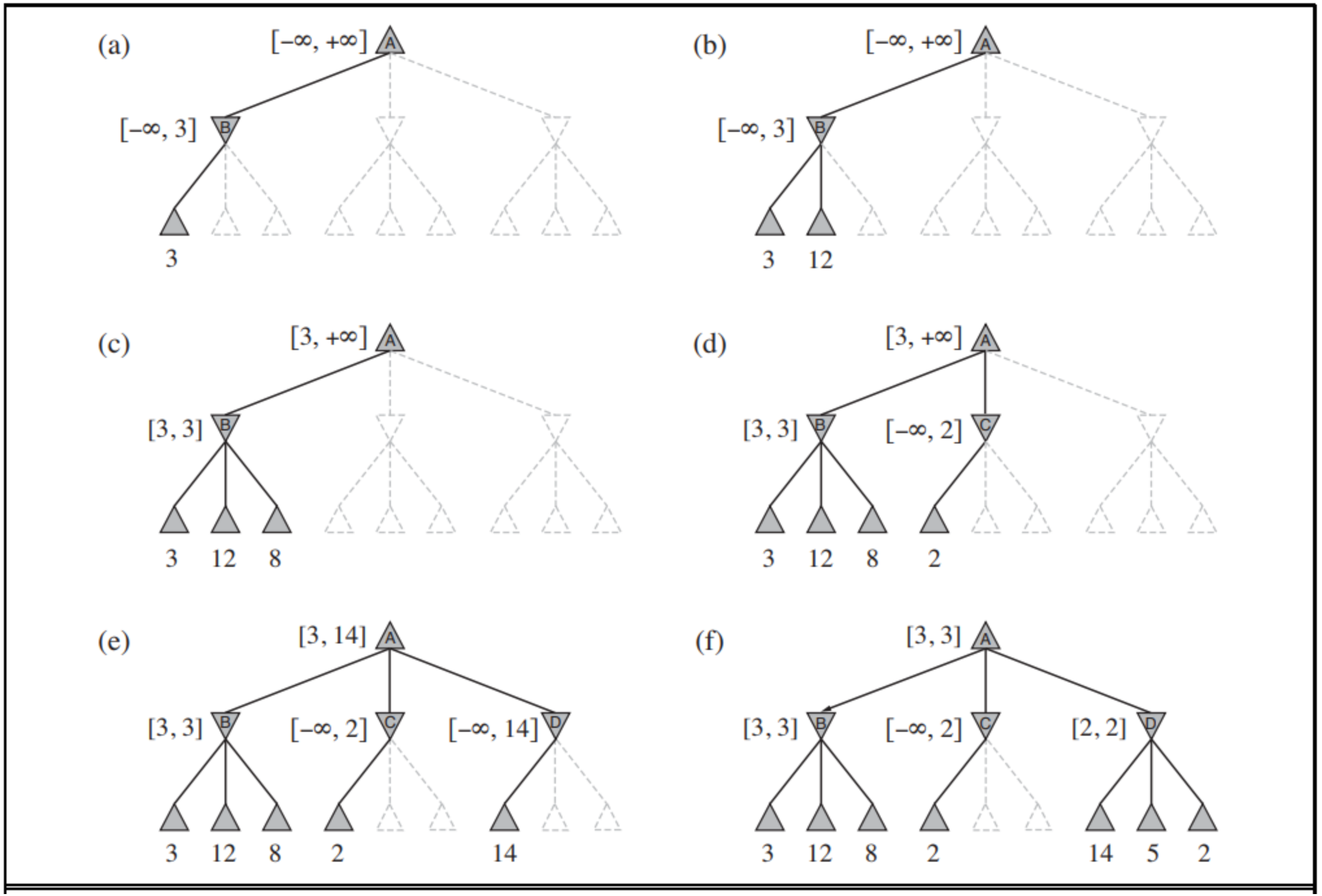


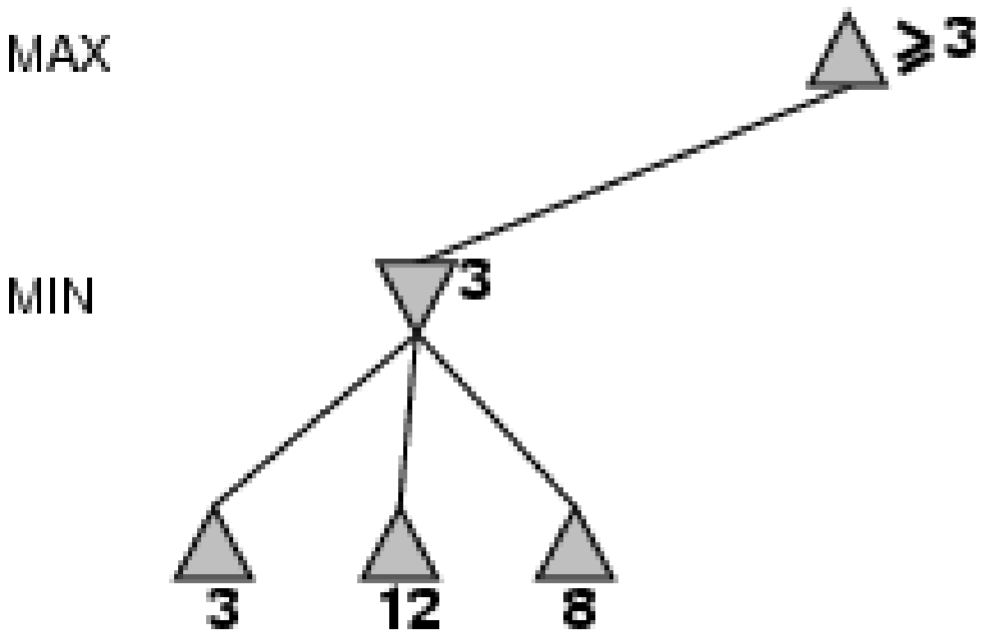
Figure 5.4 The first three plies of a game tree with three players (A, B, C). Each node is labeled with values from the viewpoint of each player. The best move is marked at the root.

Alpha-Beta Pruning

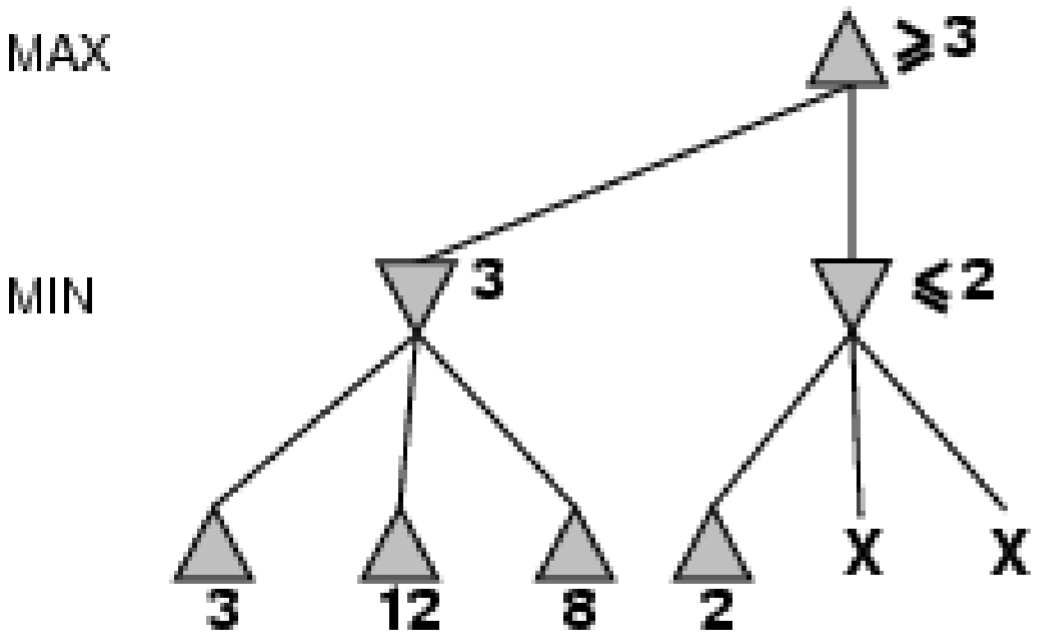
- Eliminating a branch without consideration is called pruning
- A way to improve the performance of Minimax procedure
- Basic Idea:
  - “If you have an idea which is surely bad, don’t take the time to see how awful it is” – Pat Winston


(a)
[−∞, +∞]
A
[−∞, 3]
B
3
(b)
[−∞, +∞]
A
[−∞, 3]
B
3
12
(c)
[3, +∞]
A
[3, 3]
B
3
12
8
(d)
[3, +∞]
A
[3, 3]
B
[−∞, 2]
C
3
12
8
2
(e)
[3, 14]
A
[3, 3]
B
[−∞, 2]
C
[−∞, 14]
D
3
12
8
2
14
(f)
[3, 3]
A
[3, 3]
B
[−∞, 2]
C
[2, 2]
D
3
12
8
2
14
5
2

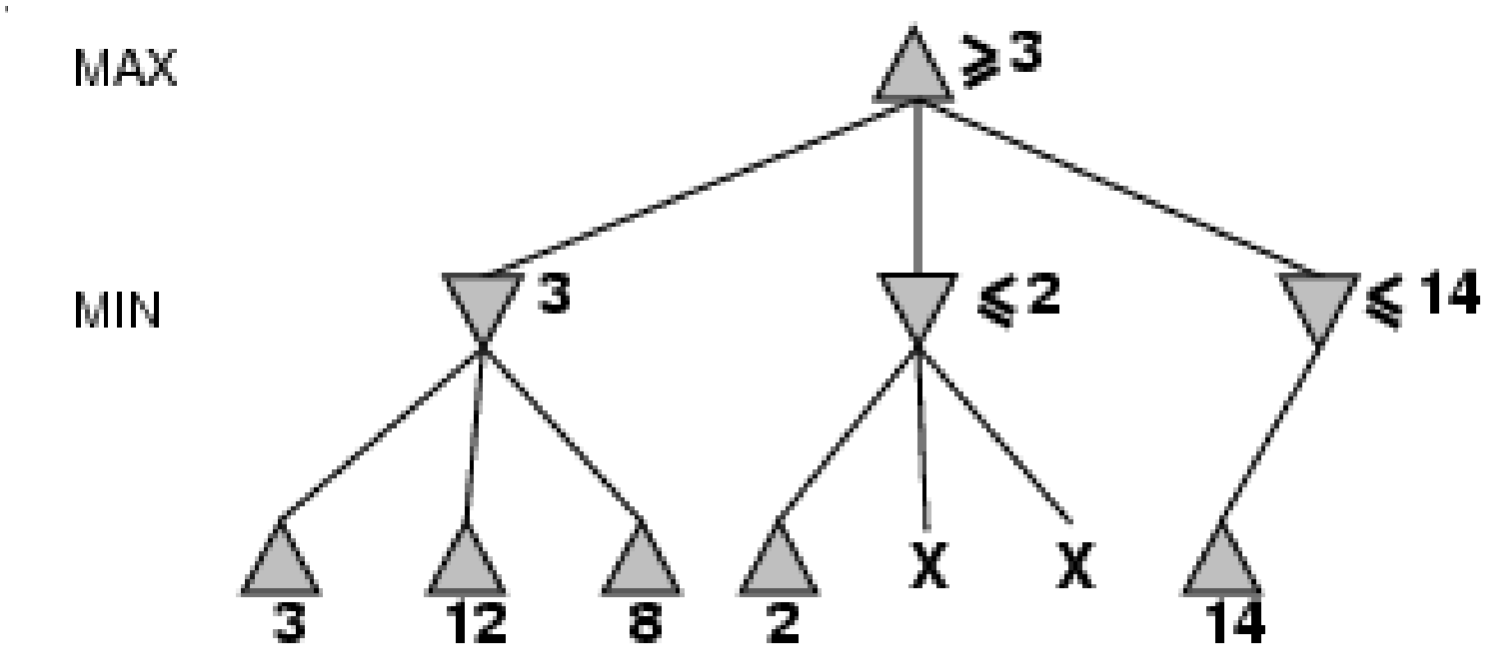
αβ Pruning example



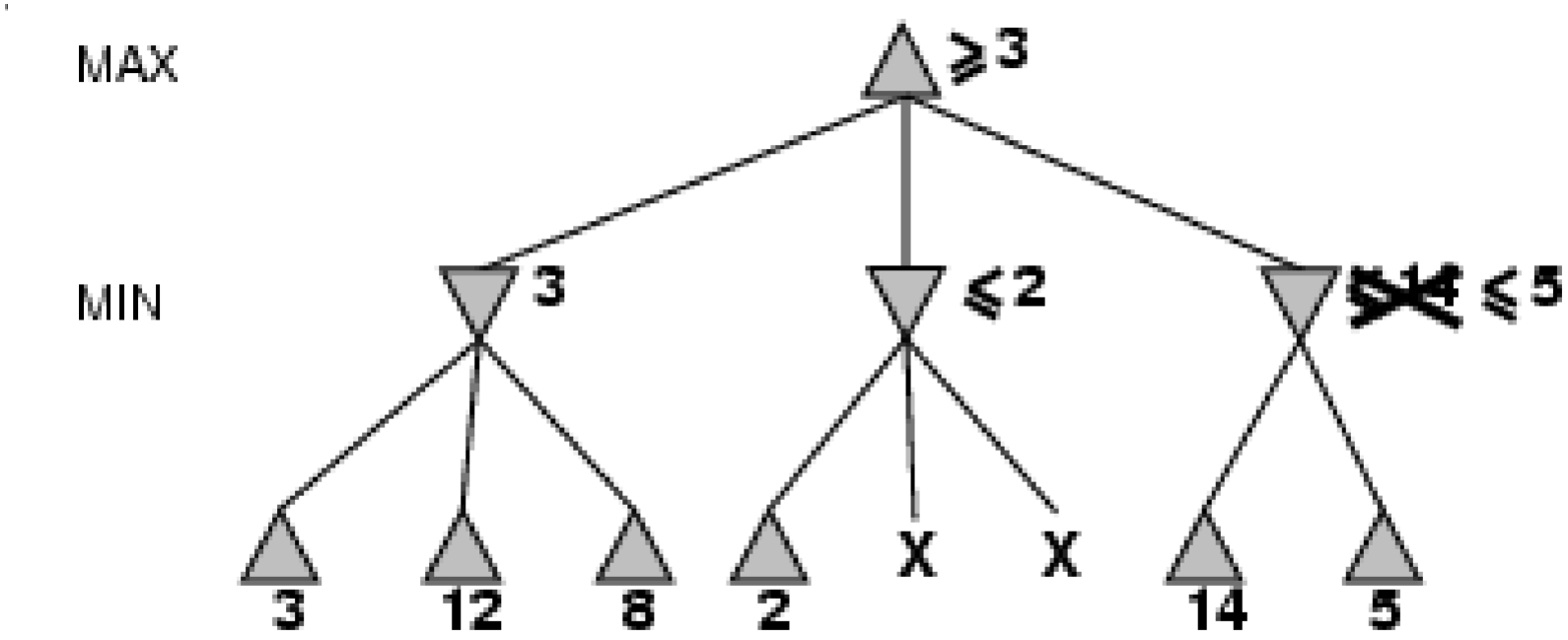
αβ pruning example



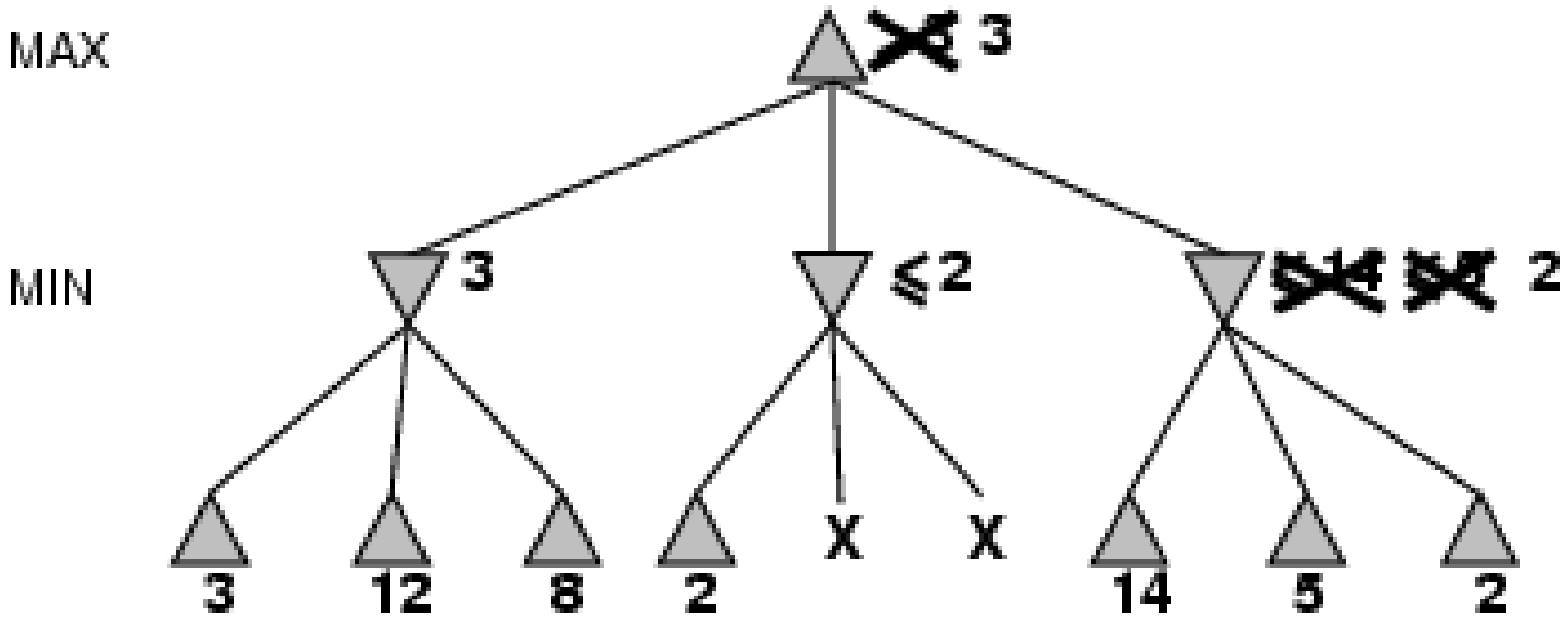
αβ pruning example

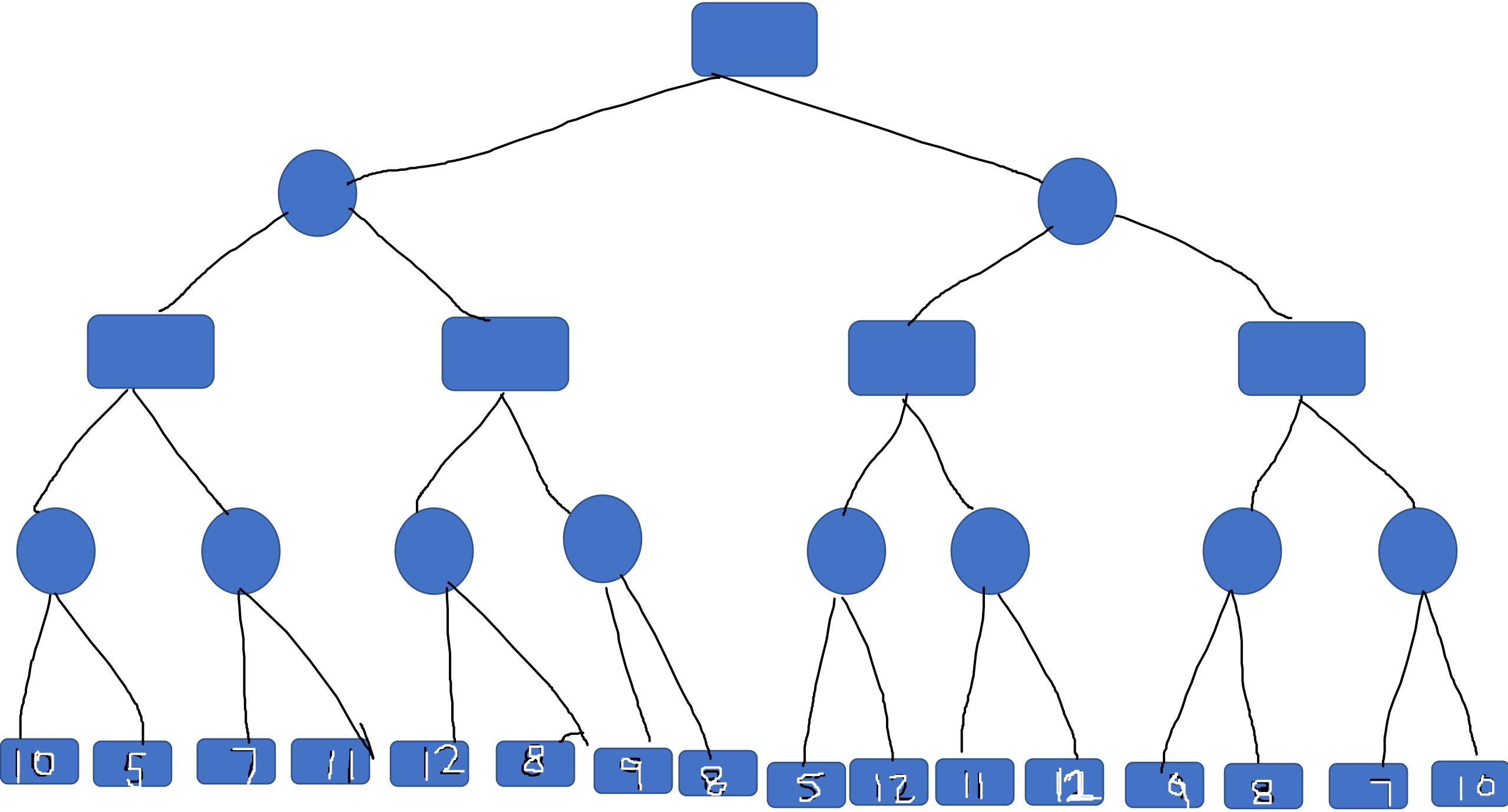


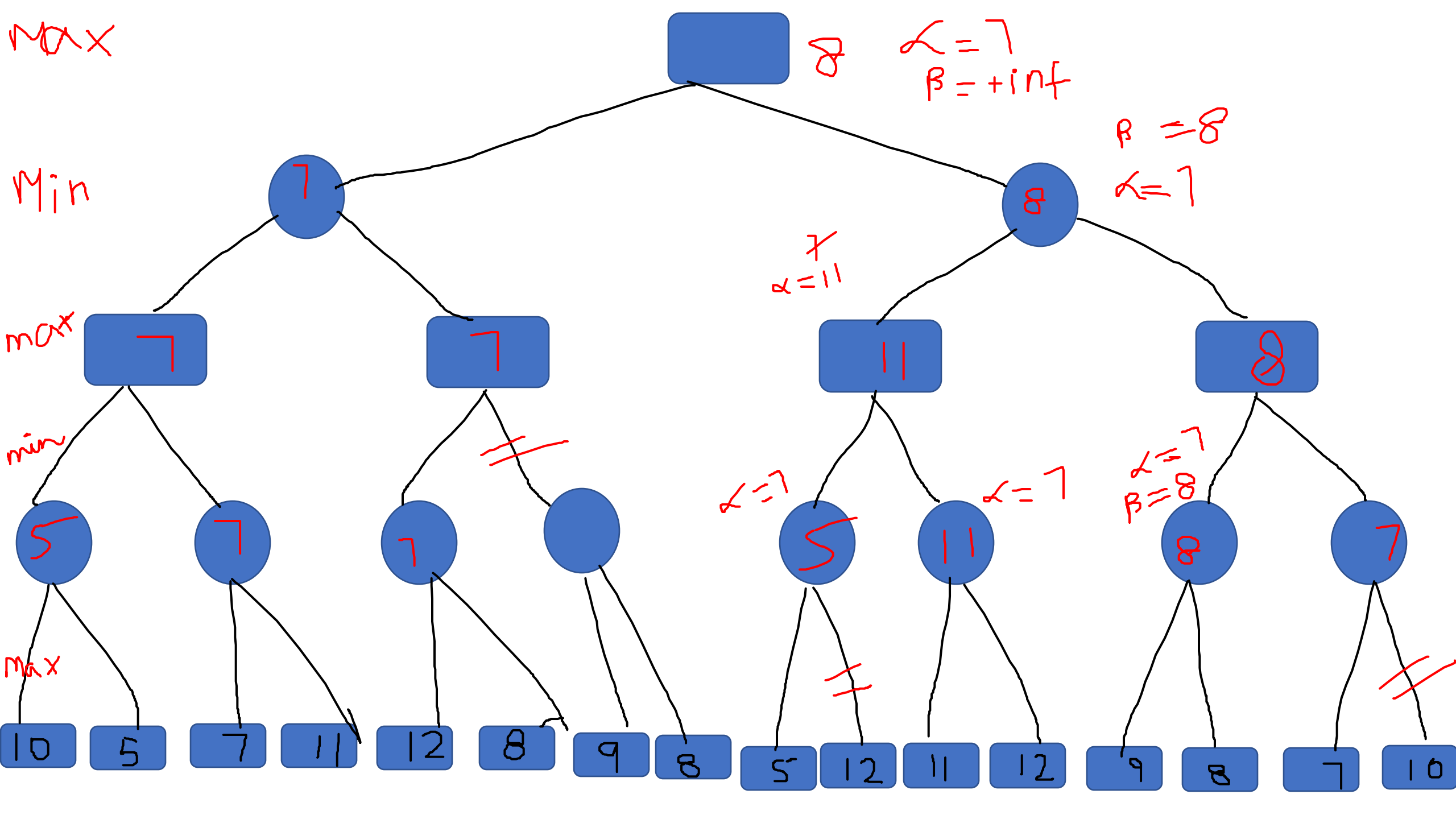
αβ pruning example



αβ pruning example




10
5
7
11
12
8
9
8
5
12
11
11
9
8
7
10


Max
8
α = 7
β = +inf
Min
7
8
β = 8
α = 7
max
7
7
11
8
α = 11
min
5
7
7
α = 7
α = 7
α = 7
β = 8
8
7
Max
10
5
7
11
12
8
9
8
5
12
11
12
9
8
7
10

Properties of αβ

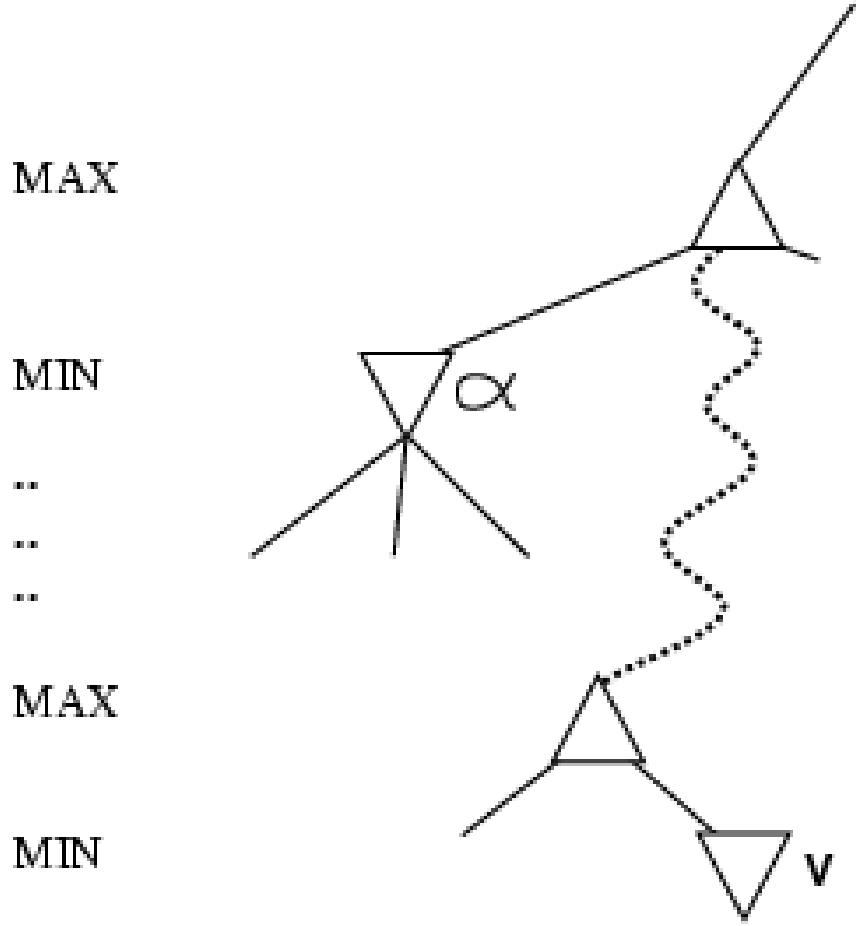
- Pruning does not affect final result
- Good move ordering improves effectiveness of pruning
- With "perfect ordering," time complexity = $O(b^{d/2})$ instead of $O(b^m)$

  -- doubles depth of search-as it can look ahead twice as compared to minimax in same amount of time

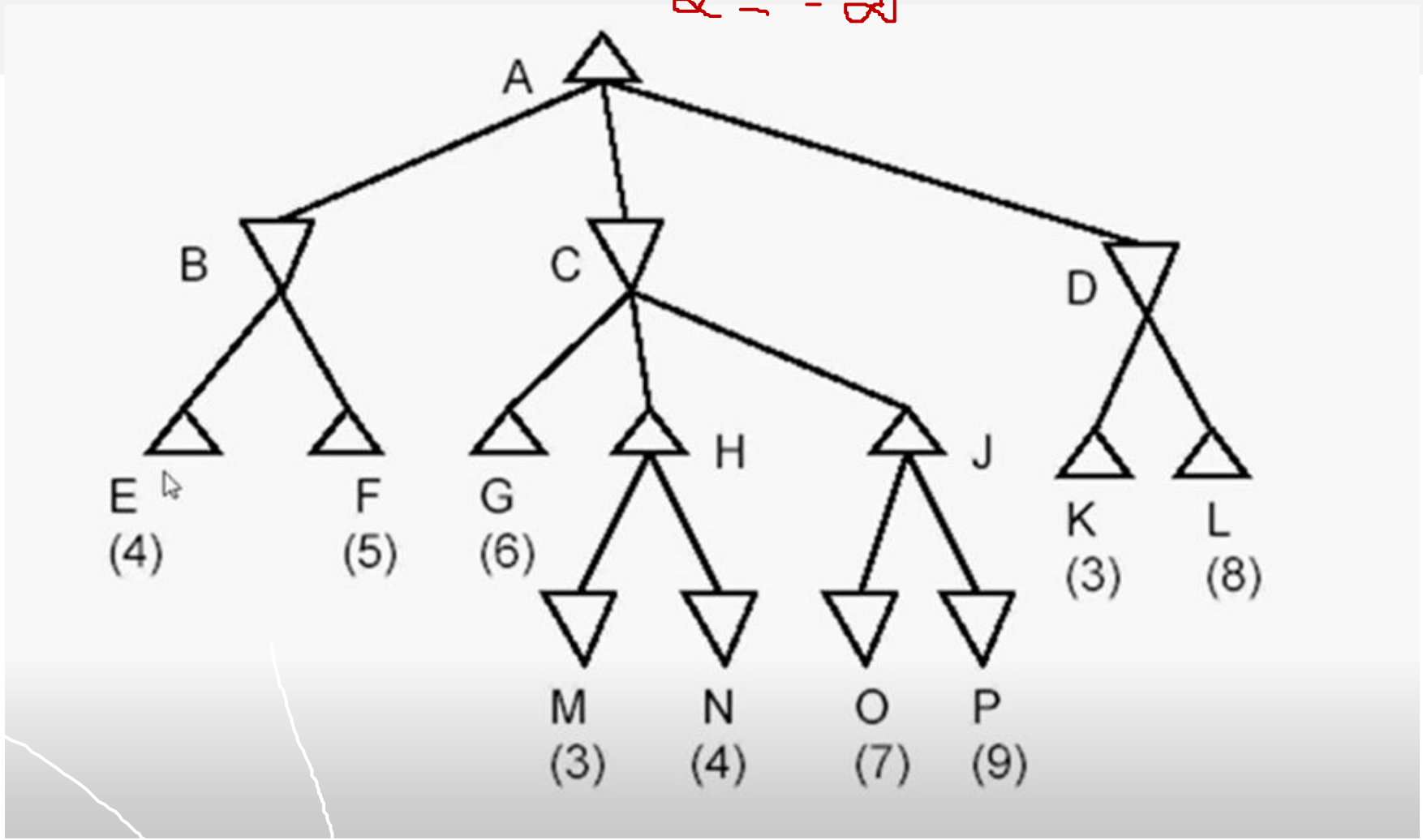
  -- If successors are examined in random order, then time complexity=$O(b^{3d/4})$

Why is it called αβ ?

- α = the value of the best (i.e., highest-value) choice we have found so far at any choice point along the path for MAX.
- β = the value of the best (i.e., lowest-value) choice we have found so far at any choice point along the path for MIN.

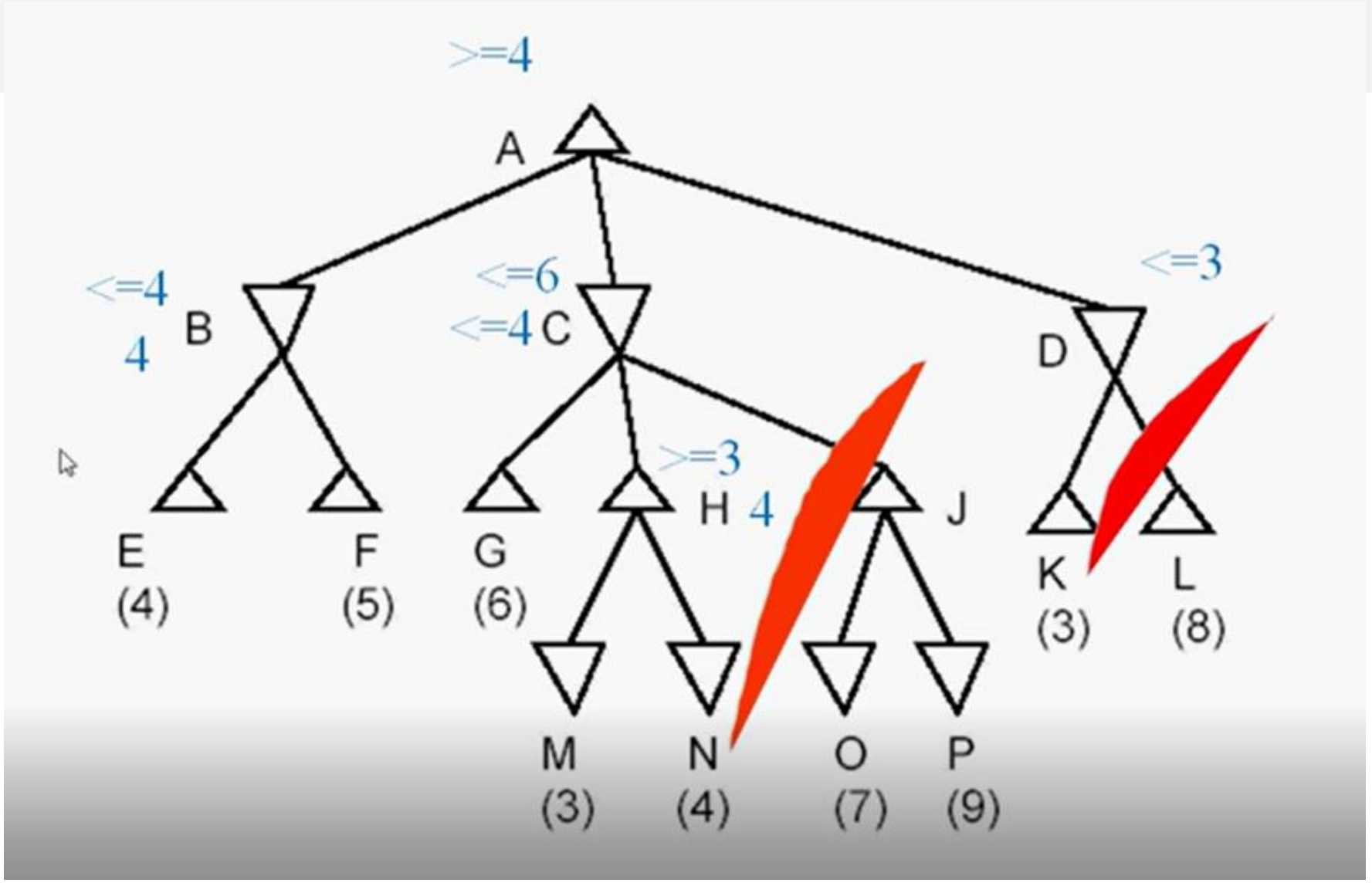


Another Example



https://www.youtube.com/watch?v=Ewh-rF7KSEg

Another Example



The αβ algorithm

```
function ALPHA-BETA-SEARCH(state) returns an action
  v ← MAX-VALUE(state, −∞, +∞)
  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 ← −∞
  for each a in ACTIONS(state) do
    v ← MAX(v, MIN-VALUE(RESULT(s,a), α, β))
    if v ≥ β then return v
    α ← MAX(α, v)
  return v

function MIN-VALUE(state, α, β) 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) , α, β))
    if v ≤ α then return v
    β ← MIN(β, v)
  return v
```

Figure 5.7 The alpha–beta search algorithm. Notice that these routines are the same as the MINIMAX functions in Figure 5.3, except for the two lines in each of MIN-VALUE and MAX-VALUE that maintain α and β (and the bookkeeping to pass these parameters along).

The αβ algorithm

```
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
```

Imperfect, Real Time Decisions

- Minimax algorithm generates entire game search space
- Alpha-Beta algorithm allows us to prune large part of it
- Usually, it is not feasible to consider the whole game tree (even with alpha–beta), so *we need to cut the search off at some point and apply a heuristic evaluation function* that estimates the utility of a state.
- Shannon's 1950 paper, Programming a computer for playing a chess
  - Programs should cut-off the search earlier-Evaluation Function
  - Utility is replaced with heuristic function
  - Terminal test is replaced by cut-off test

Resource limits

Standard approach:

- cutoff test: A cutoff test is a function that takes in a node in the game tree and **returns true if the search should be terminated at that node**, and false otherwise. The cutoff test can be based on various criteria such as depth of the search, time limit, or any other heuristic measure.

  e.g., depth limit (perhaps add quiescence search: defer evaluation until the position is stable enough)

Resource limits

Standard approach:

- evaluation function

  = estimate of the expected utility of the game from a given position

  The evaluation function should take a game **state** as input and return a score that represents how good that state is for the current player. The score should be higher if the state is more favorable for the current player, and lower if it is less favorable.

  - The **evaluation function** is used to estimate the value of a leaf node in the game tree, while the cut off test is used to determine when to stop searching down a particular branch of the tree.

Resource limits (**Stochastic games**)

- Horizon effect: The horizon effect is more difficult to eliminate. It arises when the program is facing an opponent's move that causes serious damage and is ultimately unavoidable, but can be temporarily avoided by delaying tactics.

- Singular Extension: One strategy to mitigate the horizon effect is the singular extension, a move that is "clearly better" than all other moves in a given position. Once discovered anywhere in the tree in the course of a search, this singular move is remembered. When the search reaches the normal depth limit, the algorithm checks to see if the singular extension is a legal move; if it is, the algorithm allows the move to be considered.

- Forward Pruning: Alpha–beta search prunes any node that is provably outside the current (α, β) window. PROBCUT also prunes nodes that are probably outside the window.

Evaluation functions

- For chess, typically linear weighted sum of features or Expected Value

  $Eval(s) = w_1\ f_1(s) + w_2\ f_2(s) + \ldots + w_n f_n(s)$

  - Pawn=1, bishop/knight=3, rook=5, queen=9
  - $w_1 = 9$ with $f_1(s) =$ (number of white queens) – (number of black queens), etc.
- It tells us about the "desirability" of the current state

proportion of states with each outcome. For example, suppose our experience suggests that 72% of the states encountered in the two-pawns vs. one-pawn category lead to a win (utility +1); 20% to a loss (0), and 8% to a draw (1/2). Then a reasonable evaluation for states in the category is the **expected value**: $(0.72 \times +1) + (0.20 \times 0) + (0.08 \times 1/2) = 0.76$. In

Cutting off search

MinimaxCutoff is identical to MinimaxValue

except

1. Terminal? is replaced by Cutoff?
2. Utility is replaced by Eval

Does it work in practice?

$b^m = 10^6$, b=35 -> m=4

4 ply lookahead is a hopeless chess player!

- 4-ply ≈ human novice
- 8-ply ≈ typical PC, human master
- 12-ply ≈ Deep Blue, Kasparov

Deterministic games in practice

- Checkers: Chinook ended 40-year-reign of human world champion Marion Tinsley in 1994. Used a pre-computed endgame database defining perfect play for all positions involving 8 or fewer pieces on the board, a total of 444 billion positions
- Chess: Deep Blue defeated human world champion Garry Kasparov in a six-game match in 1997. Deep Blue searches 200 million positions per second, uses very sophisticated evaluation, and undisclosed methods for extending some lines of search up to 40 ply
- Othello: human champions refuse to compete against computers, who are too good
- Go: human champions refuse to compete against computers, who are too bad. In go, $b > 300$, so most programs use pattern knowledge bases to suggest plausible moves.

Summary

- Games are fun to work on!
- They illustrate several important points about AI
- perfection is unattainable → must approximate
- good idea to think about what to think about

Homework

- Readings
  - CH 5 Section 5.5