unit-5

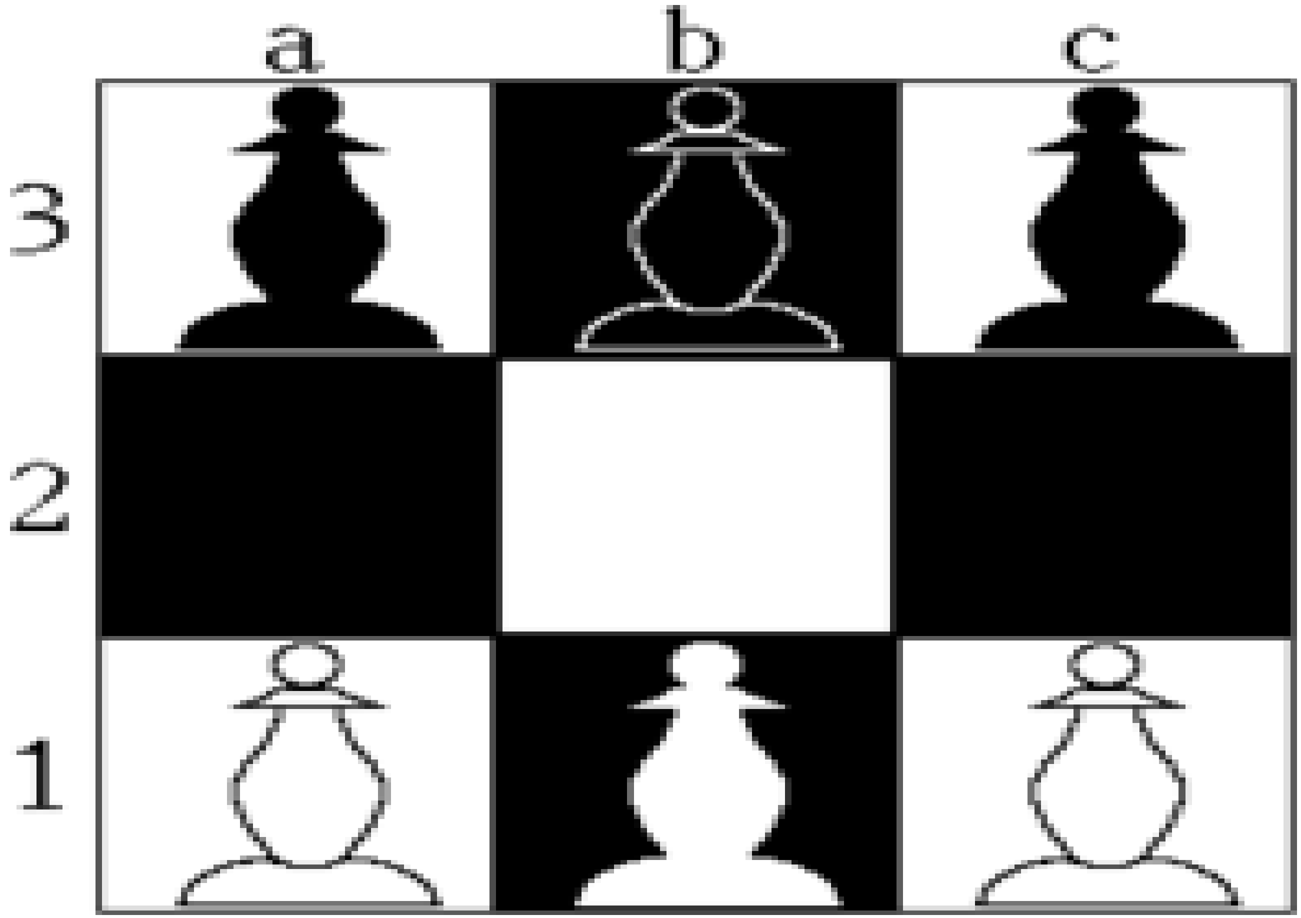
Building games with AI

Building games with AI : using search algorithms in games, combinatorial search, minmax problem, alpha-beta pruning, negamax algorithm, last coin standing game,

tic-tac-toe game, hexapawn game.

tic-tac-toe game

Hexapawn game

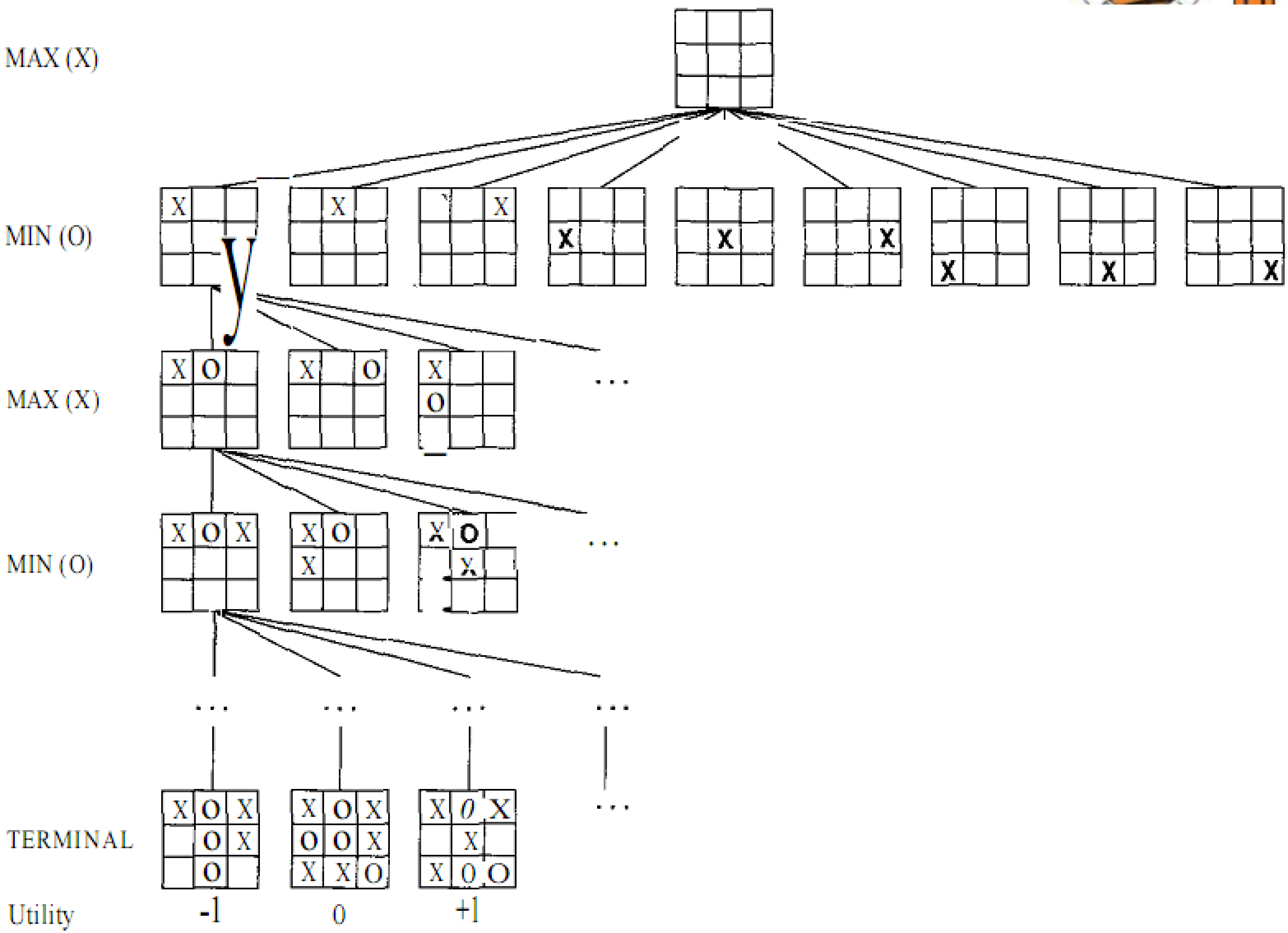


Last coin game

MiniMax Algorithm

A game can be formally defined as a kind of search problem with the following components:

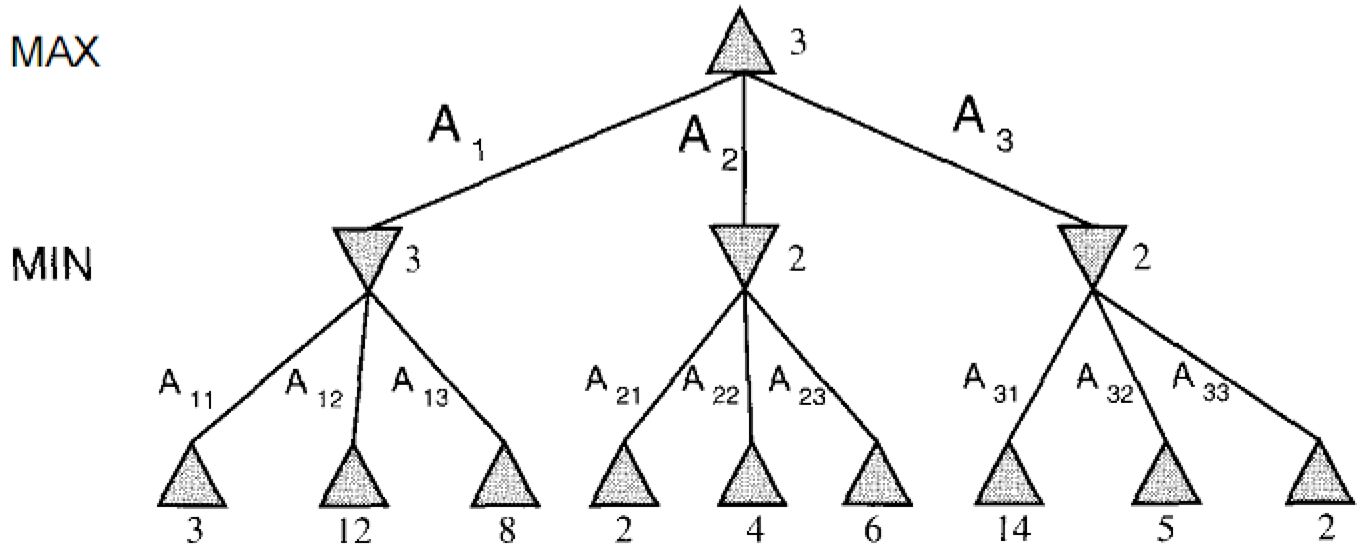
- **The initial state,** which includes the board position and an indication of whose move it is.
- **A set** of **operators,** which define the legal moves that a player can make.
- A **terminal test,** which determines when the game is over. States where the game has ended are called **terminal states.**
- A **utility function** (also called a **payoff function),** which gives a numeric value for the outcome of a game. In chess, the outcome is a win, loss, or draw, which we can represent by the values +1, −1, or 0. Some games have a wider variety of possible outcomes; for example, the payoffs in backgammon range from +192 to −192.


MAX (X)
MIN (O)
MAX (X)
MIN (O)
TERMINAL
Utility
-1
0
+1

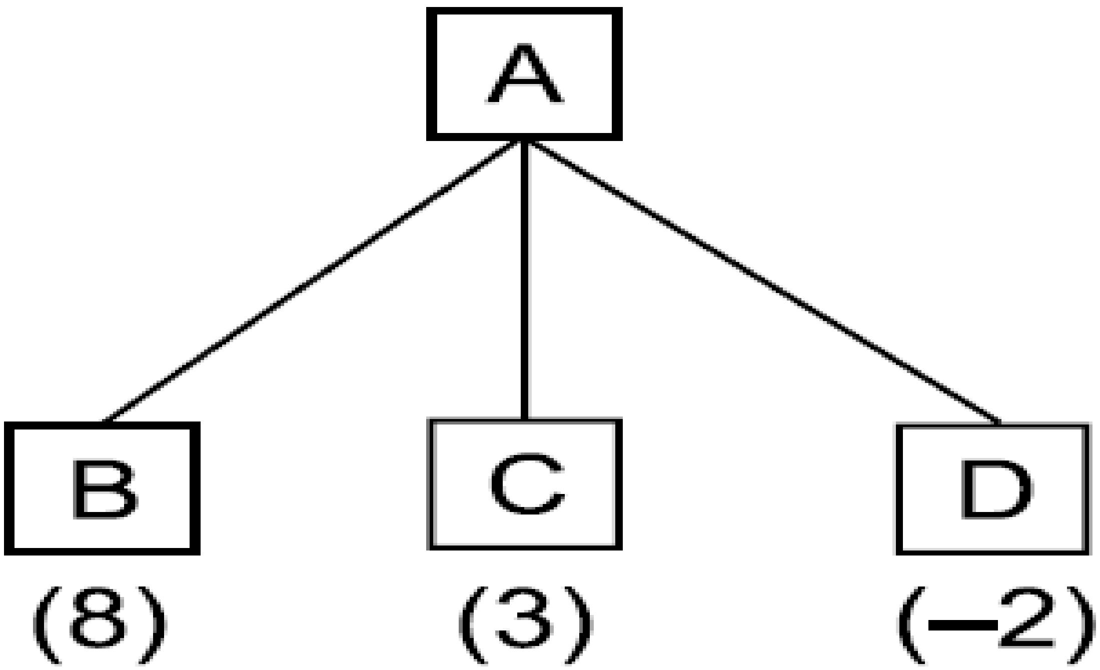
Process of MINIMAX Algorithm

The **minimax** algorithm is designed to determine the optimal strategy for MAX, and thus to decide what the best first move is. The algorithm consists of five steps:

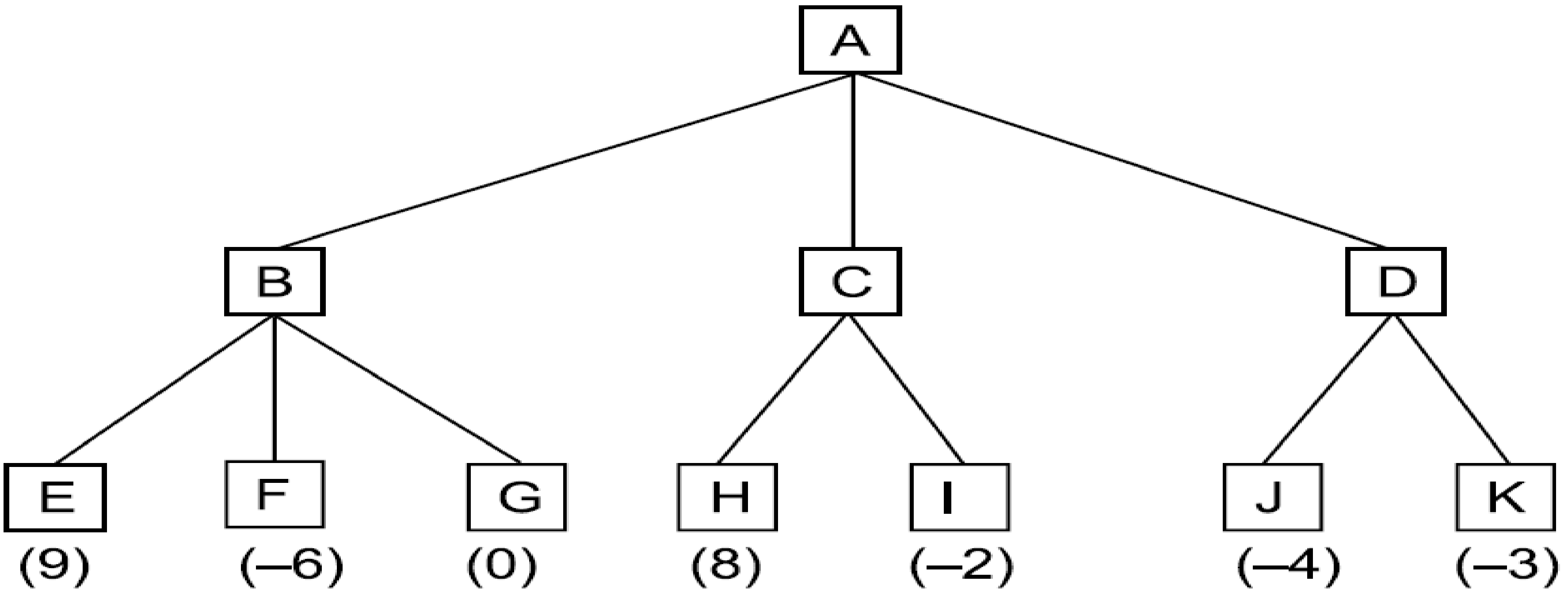
- Generate the whole game tree, all the way down to the terminal states.
- Apply the utility function to each terminal state to get its value.
- Use the utility of the terminal states to determine the utility of the nodes one level higher up in the search tree. Consider the leftmost three leaf nodes in Figure 5.2. In the V node above it, MIN has the option to move, and the best MIN can do is choose A_{11}, which leads to the minimal outcome, 3. Thus, even though the utility function is not immediately applicable to this V node, we can assign it the utility value 3, under the assumption that MIN will do the right thing. By similar reasoning, the other two V nodes are assigned the utility value 2.
- Continue backing up the values from the leaf nodes toward the root, one layer at a time.
- Eventually, the backed-up values reach the top of the tree; at that point, MAX chooses the move that leads to the highest value. In the topmost A node ofFigure 5.2, MAX has a choice of three moves that will lead to states with utility 3, 2, and 2, respectively. Thus, MAX's best opening move is AJ_1. This is called the **mininiax decision,** because it maximizes the utility under the assumption that the opponent will play perfectly to minimize it.


MAX
MIN
3
A 1
A 2
A 3
3
2
2
A 11
A 12
A 13
A 21
A 22
A 23
A 31
A 32
A 33
3
12
8
2
4
6
14
5
2

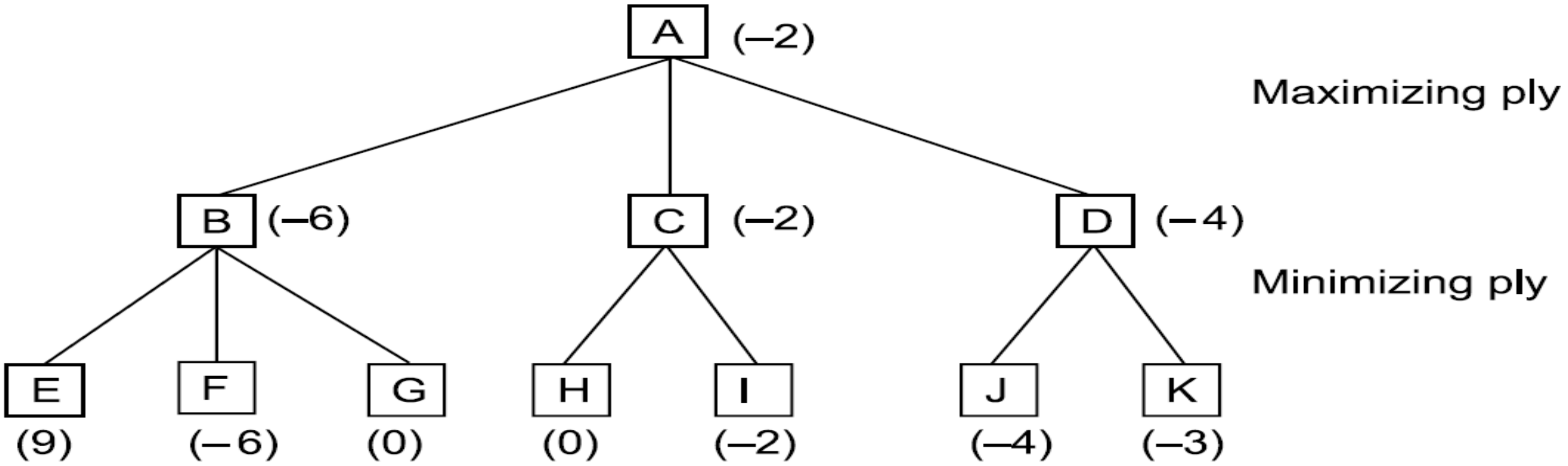
One – Ply Search



Two – Ply Search



Backing Up the Values of a Two – Ply Search

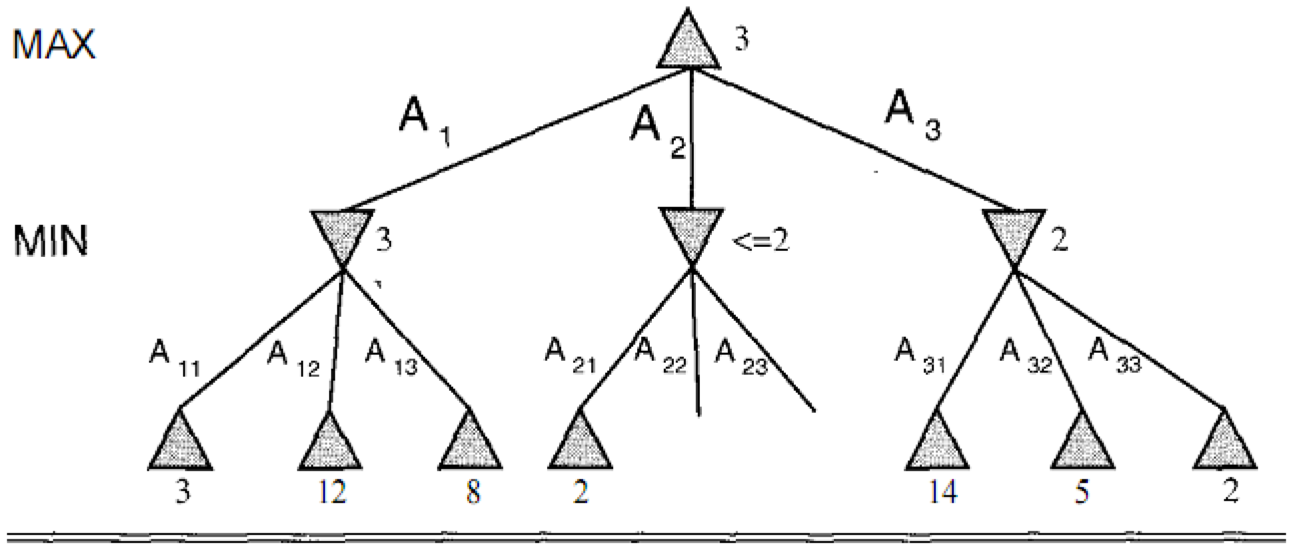


```
function MINIMAX-DECISION(game) returns an operator

  for each op in OPERATORS[game] do
      VALUE[op] — MINIMAX-VALUE(APPLY(op, game), game)
  end
  return the op with the highest VALUE[op]
```

```
function MINIMAX-VALUE(state, game) returns a utility value

  if TERMINAL-TEST[game](state) then
      return   UTILITY[game](state)
  else if MAX is to move in state then
      return the highest MINIMAX-VALUE of SUCCESSORS(state)
  else
      return the lowest MINIMAX-VALUE of SUCCESSORS(state)
```

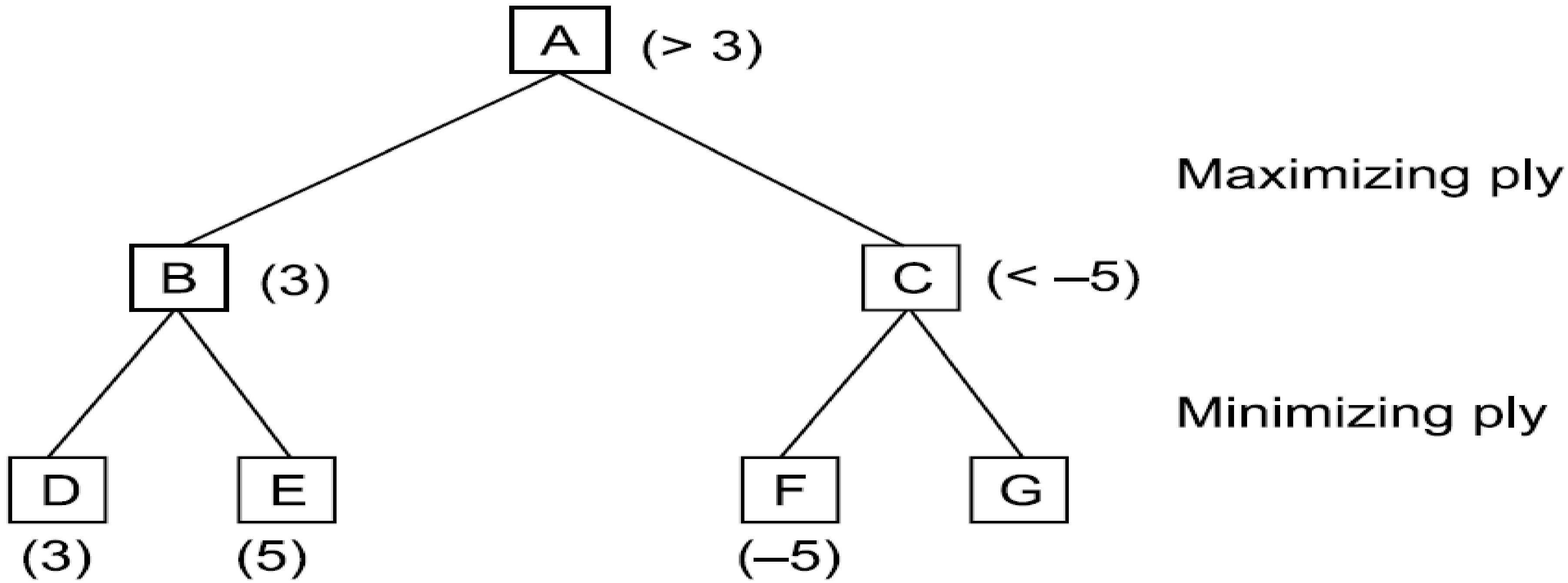
ALPHA-BETA PRUNING

Fortunately, it is possible to compute the correct minimax decision without looking at every node in the search tree. The process of eliminating a branch of the search tree from consideration without examining it is called **pruning** the search tree. The particular technique we will examine is called **alpha-beta pruning.** When applied to a standard minimax tree, it returns the same move as minimax would, but prunes away branches that cannot possibly influence the final decision.

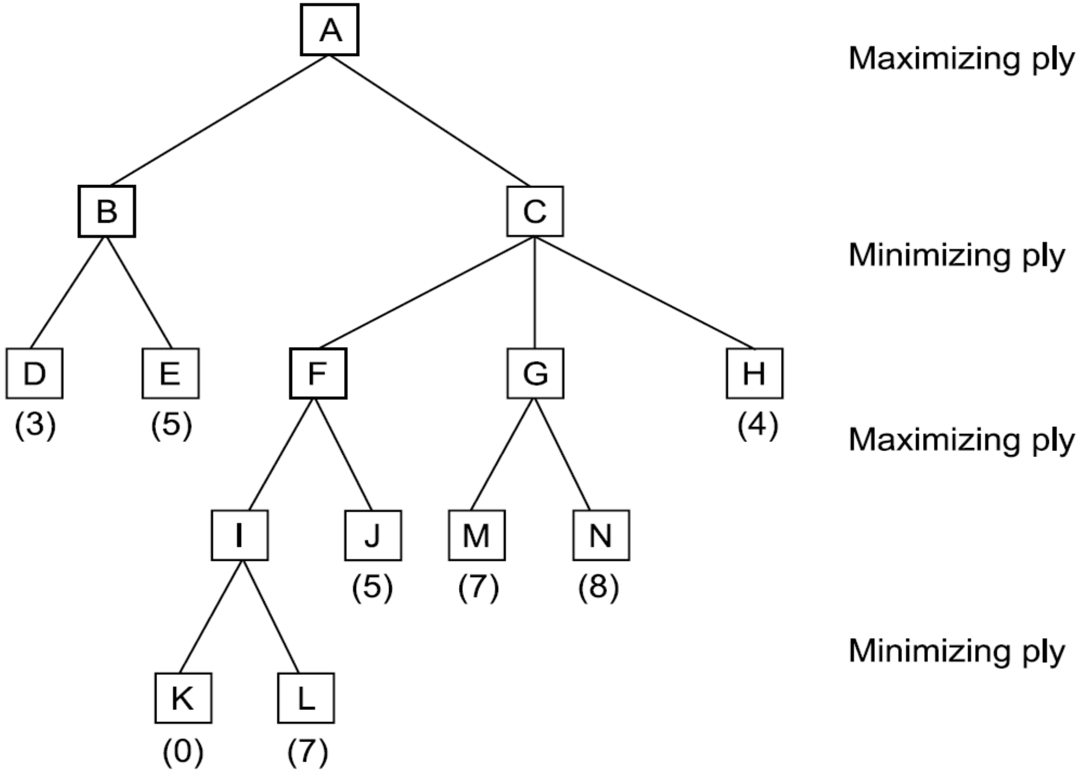


The two-ply game tree as generated by alpha-beta.

Alpha – Beta Pruning



Alpha and Beta Cutoffs



The general principle is this. Consider a node *n* somewhere in the tree (see Figure 5.7), such that Player has a choice of moving to that node. If Player has a better choice *m* either at the parent node of n, or at any choice point further up, then *n will never be reached in actual play*.

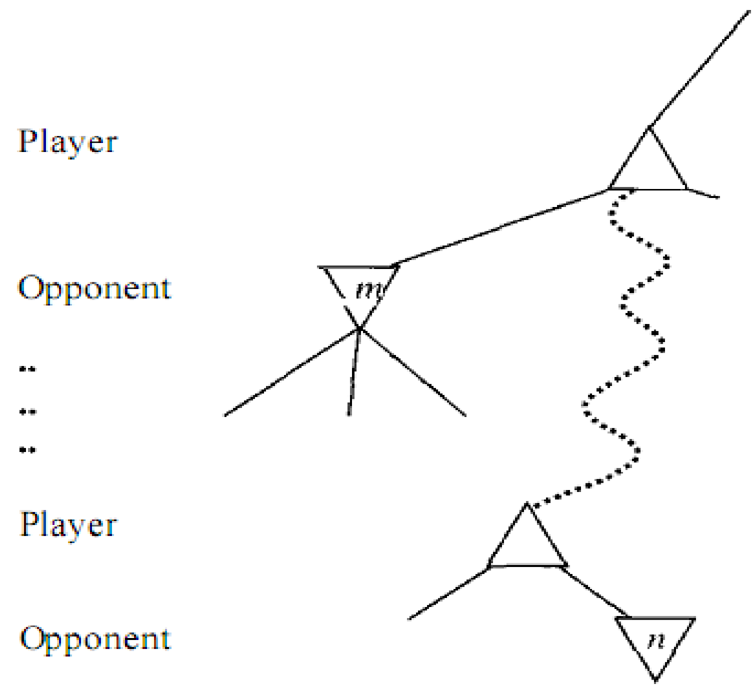
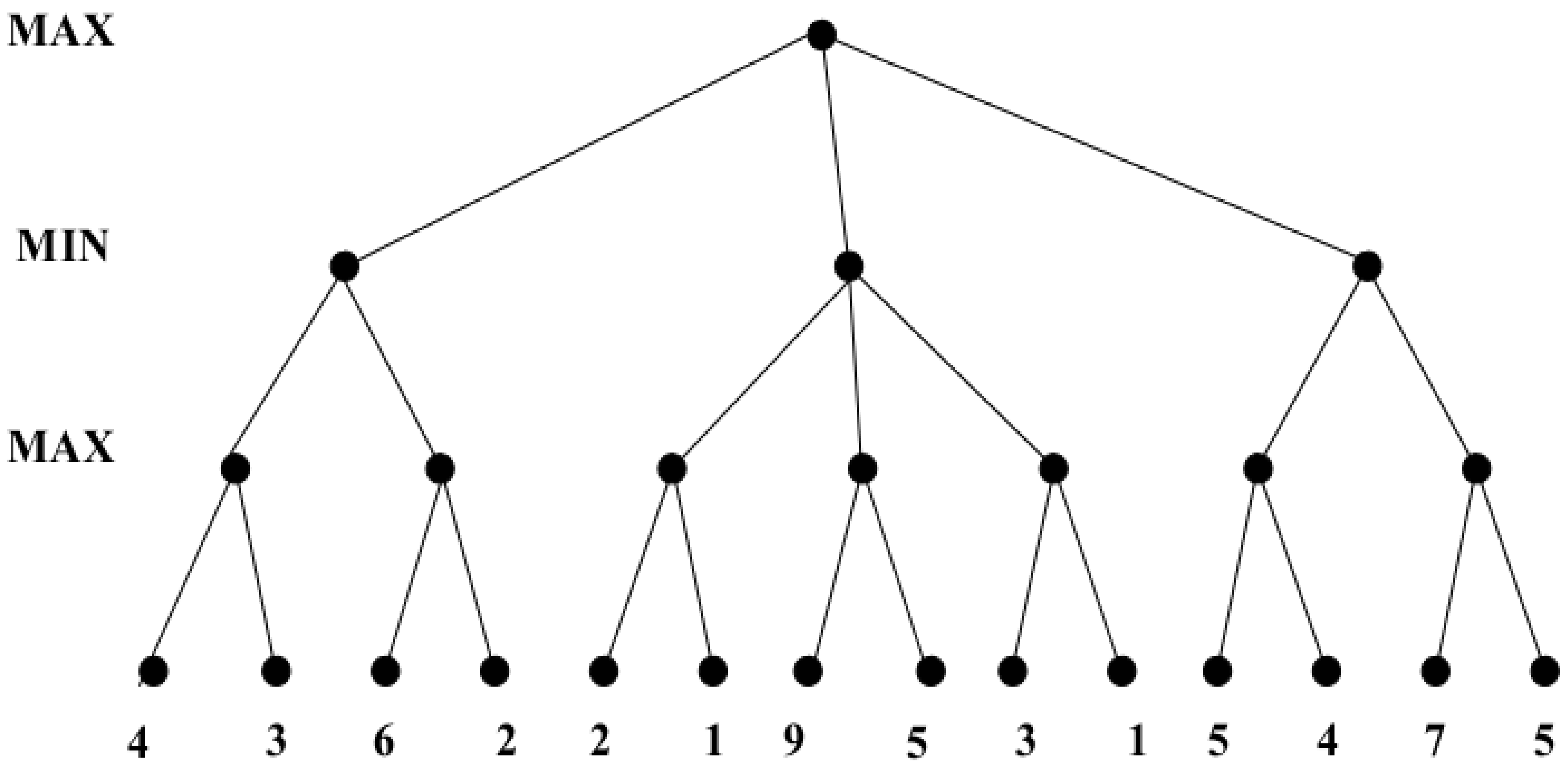


Figure 5.7 Alpha-beta pruning: the general case. If *m* is better than *n* for Player, we will never get to *n* in play.

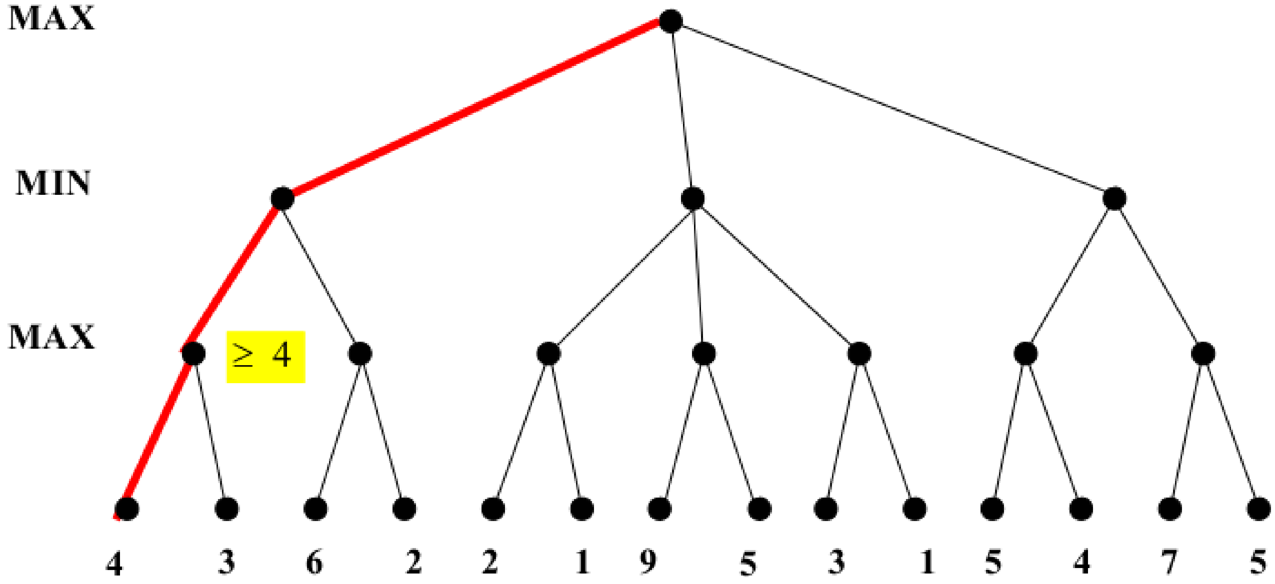
```
function MAX-VALUE(stategame, a, β) returns the minimax value of state
  inputs: state, current state in game
          game, game description
          a, the best score for MAX along the path to state
          β, the best score for MIN along the path to state

  if CUTOFF-TEST(state) then return EVAL(state)
  for each s in SUCCESSORS(state) do
      a ← MAX(α , MIN-VALUE(s, game, a, β))
      if a ≥ ft then return ft
  end
  return a
```

```
function MIN-VALUE(state, game, a, ft) returns the minimax value of state

  if CUTOFF-TEST(state) then return EVAL(state)
  for each s in SUCCESSORS(state) do
      ft — MIN( ft, MAX-VALUE(s, game, a, β))
      if ft < a then return a
  end
  return ft
```

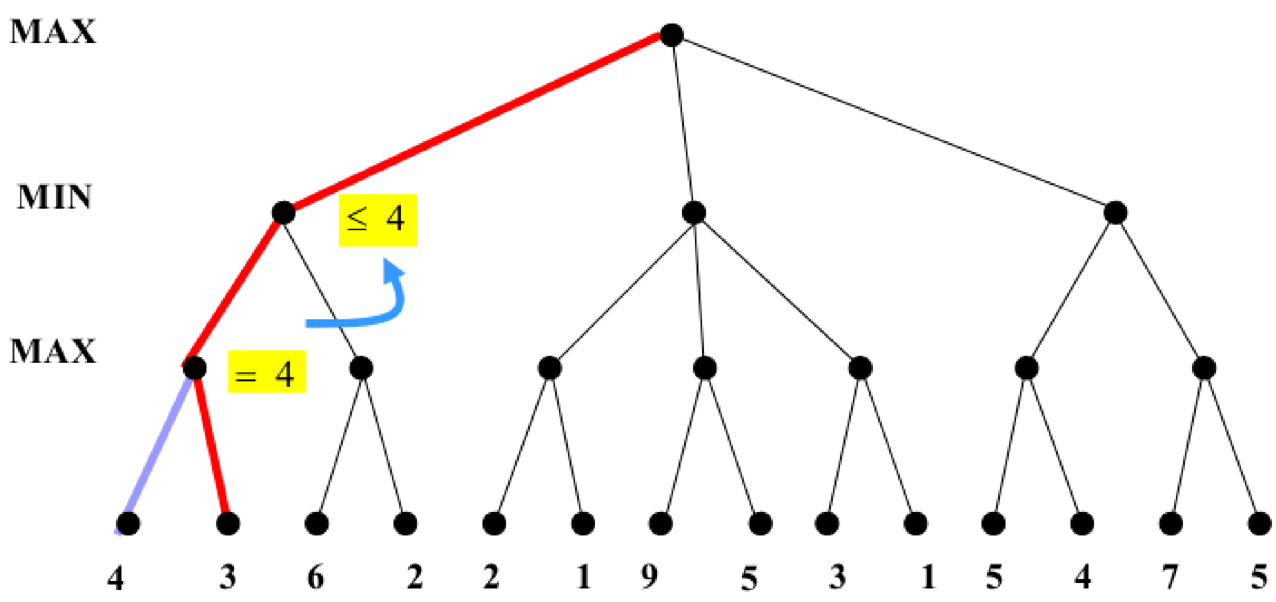
Alpha beta pruning. Example



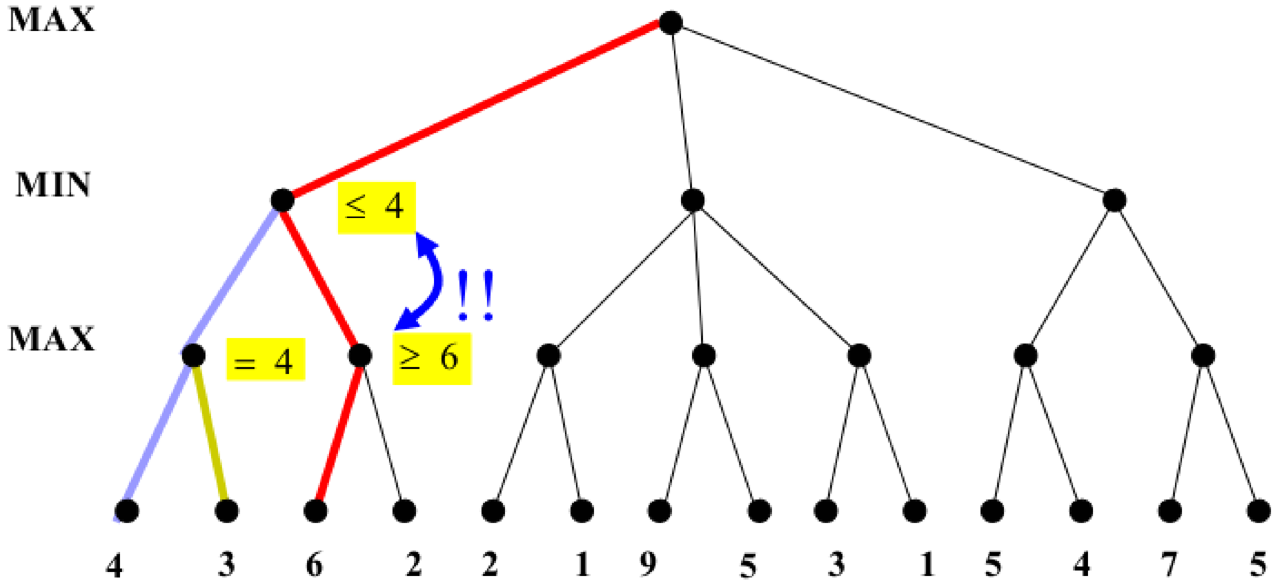
Alpha beta pruning. Example



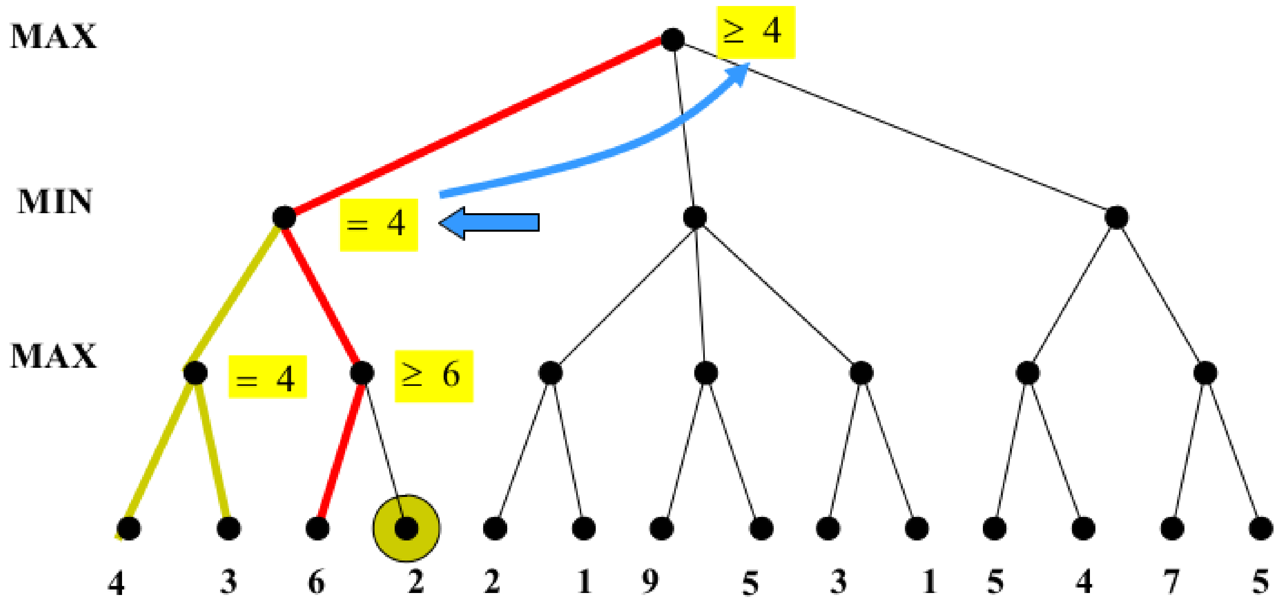
Alpha beta pruning. Example



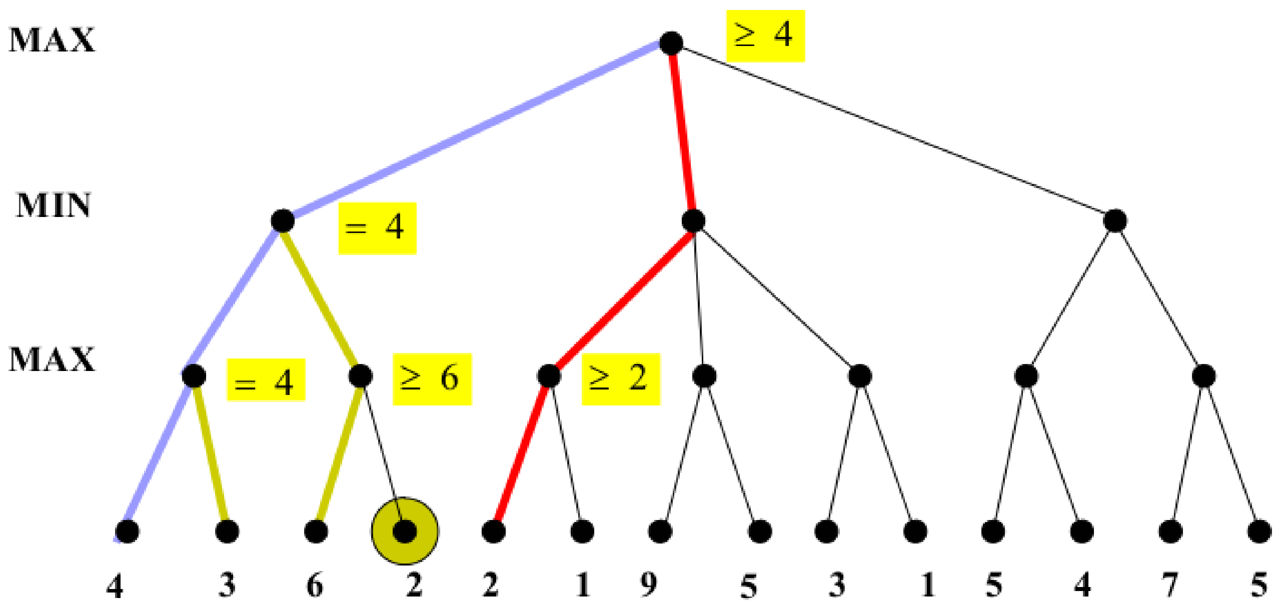
Alpha beta pruning. Example



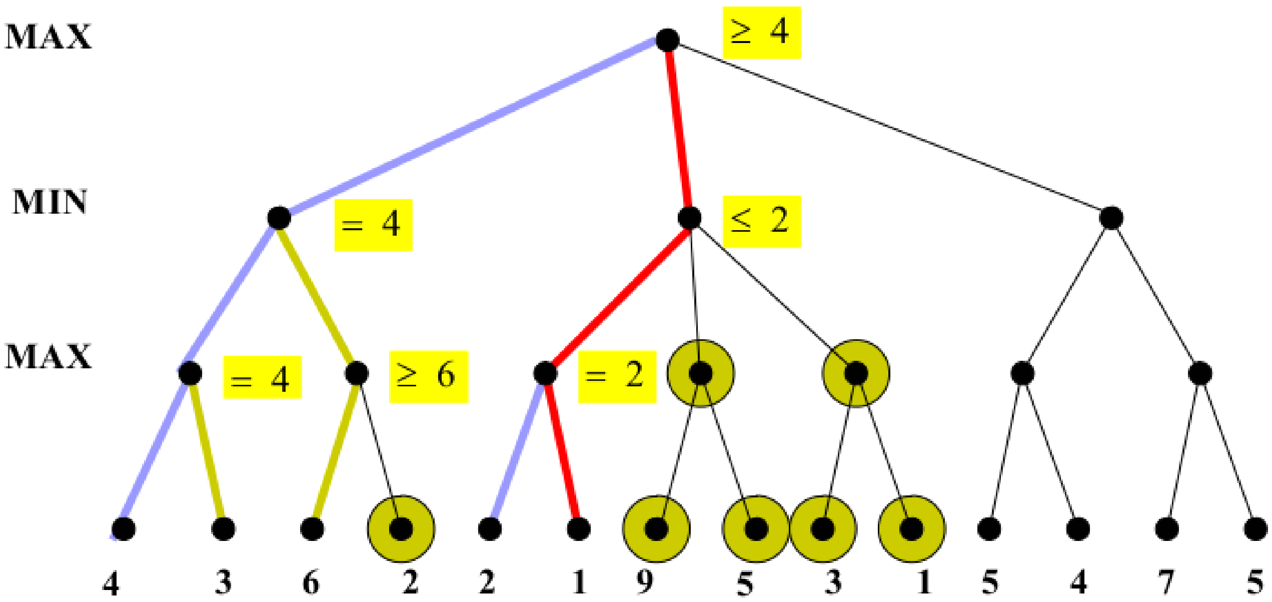
Alpha beta pruning. Example



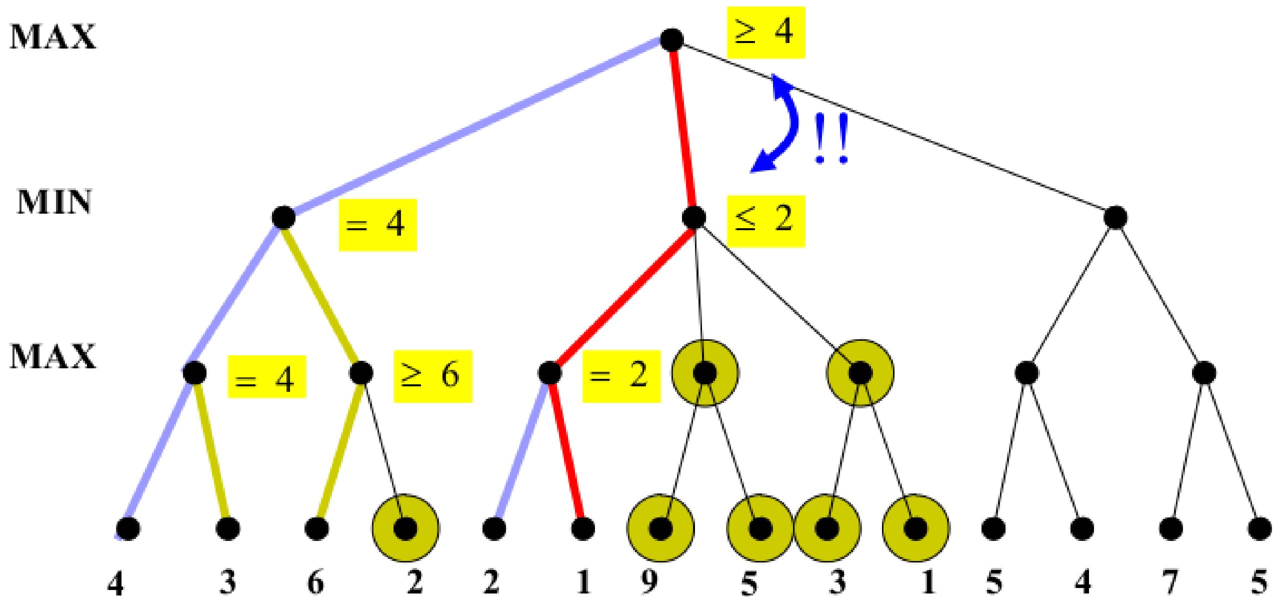
Alpha beta pruning. Example



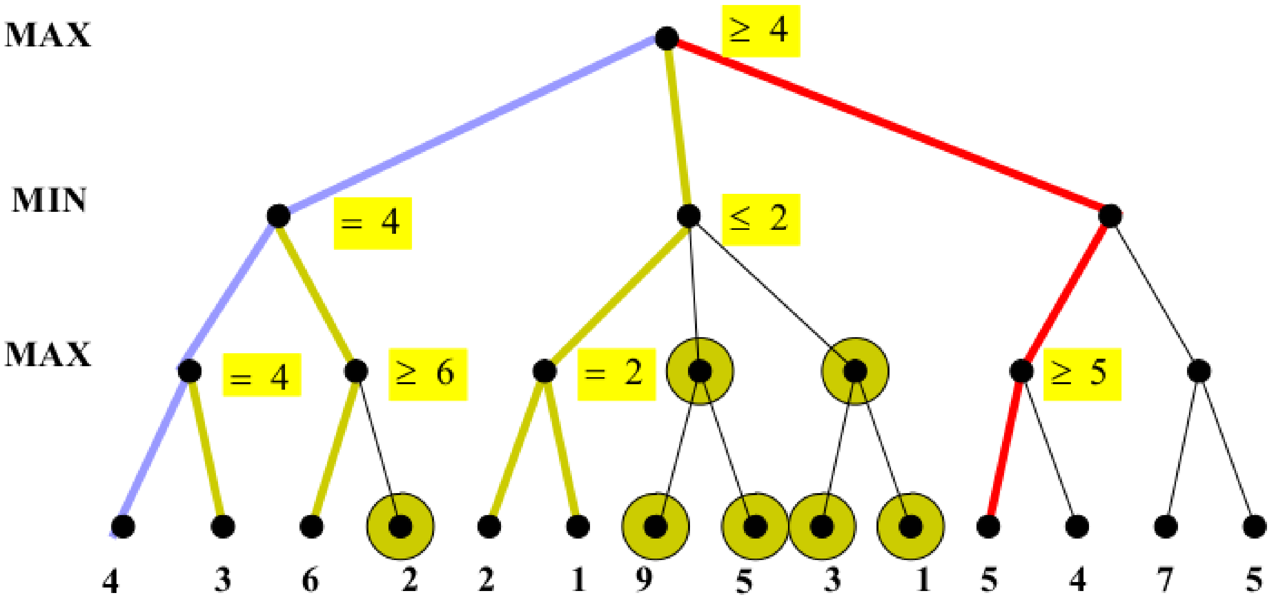
Alpha beta pruning. Example



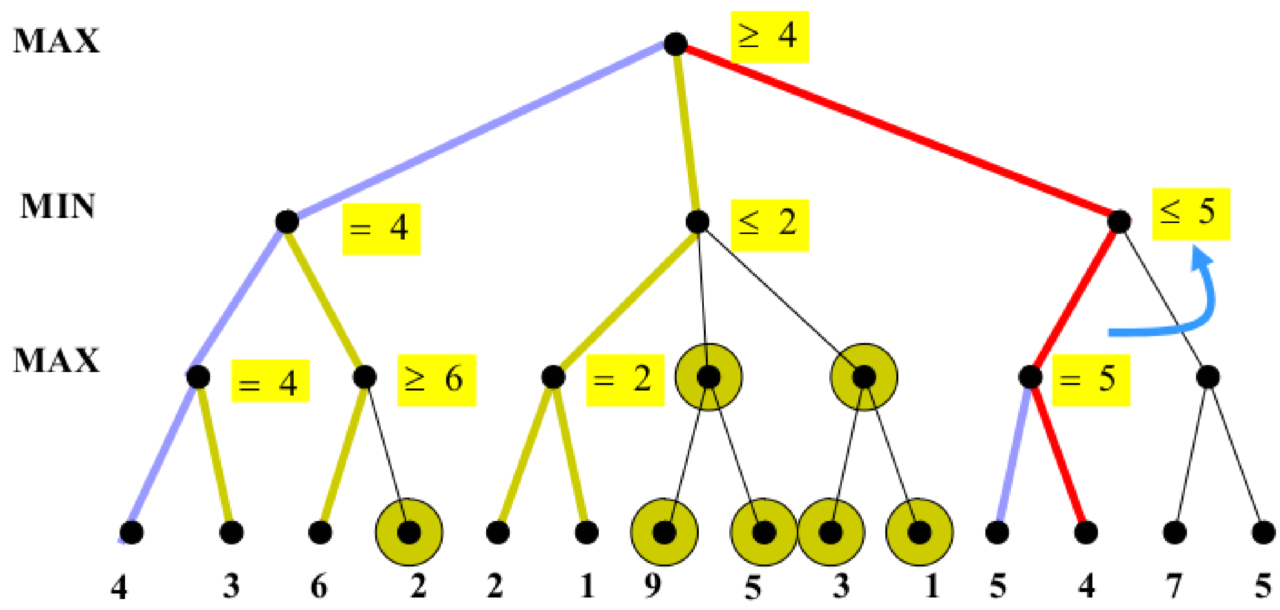
Alpha beta pruning. Example



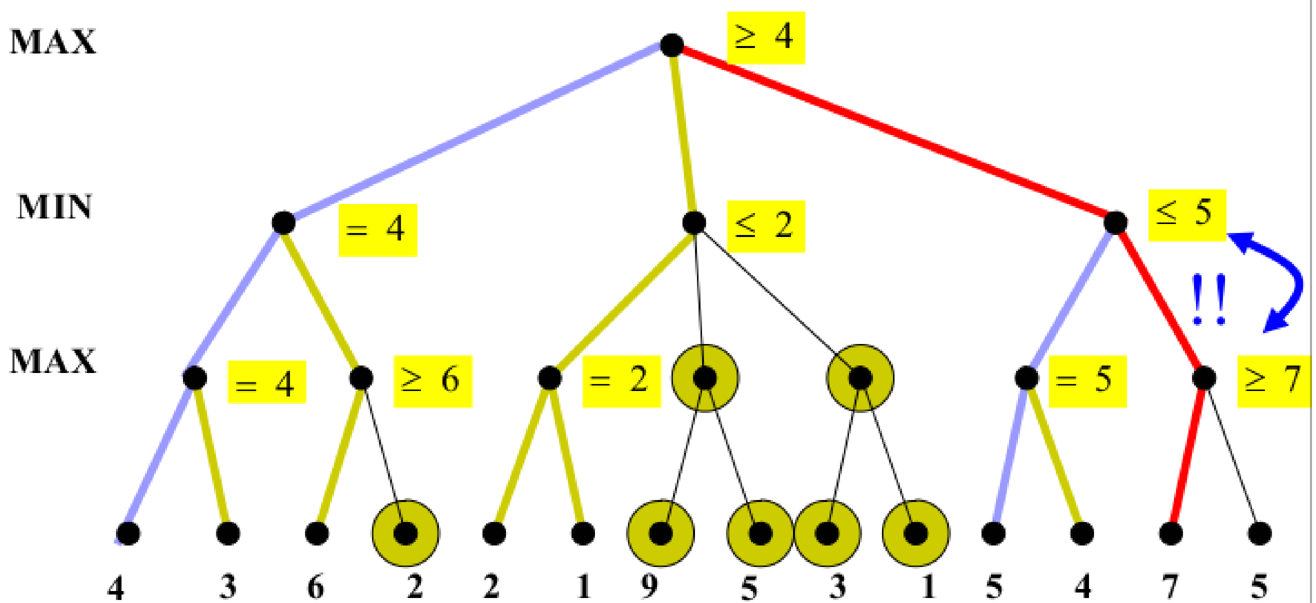
Alpha beta pruning. Example



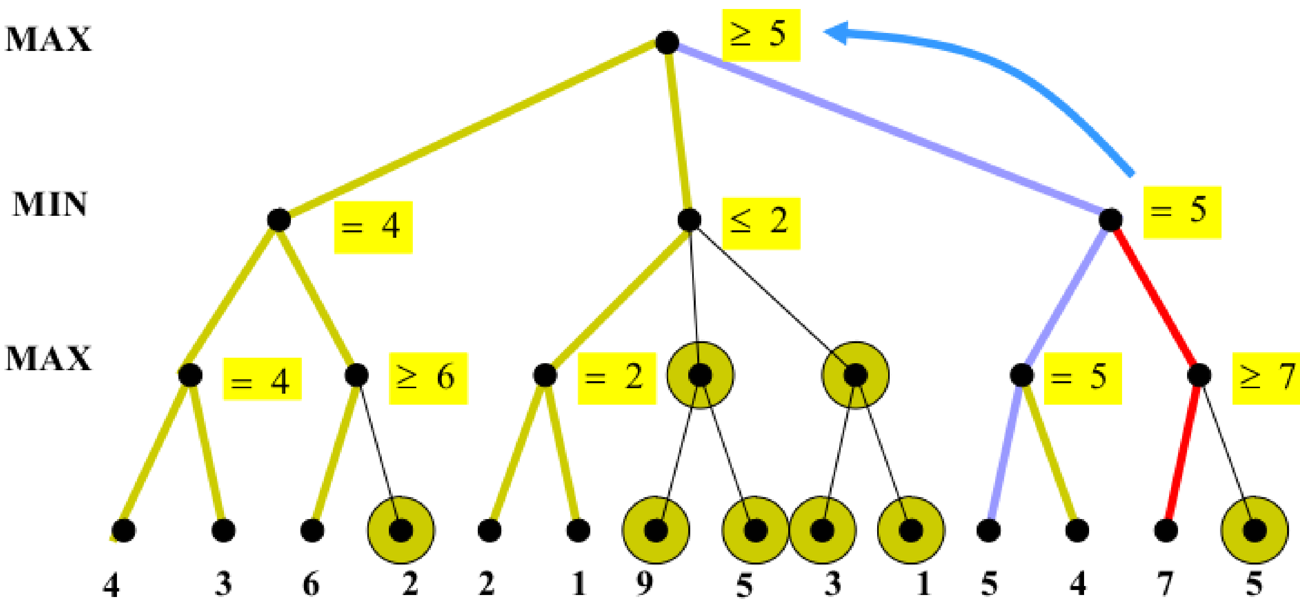
Alpha beta pruning. Example



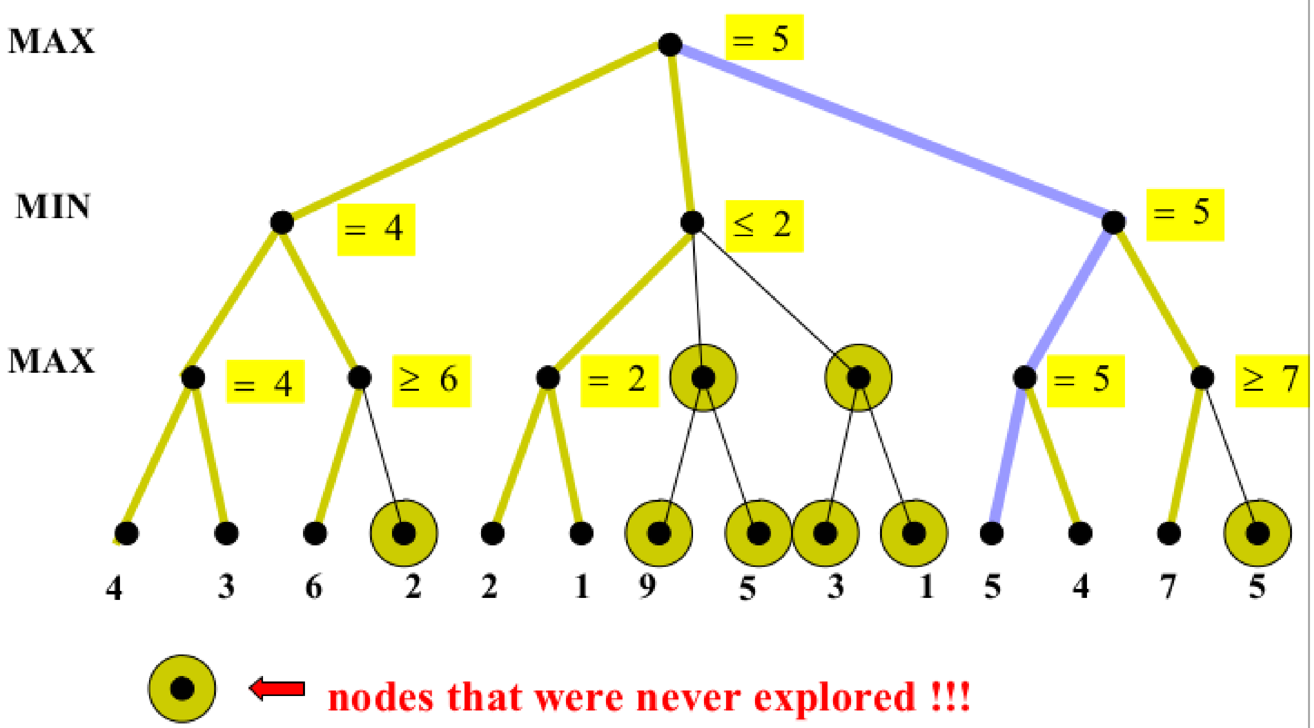
Alpha beta pruning. Example

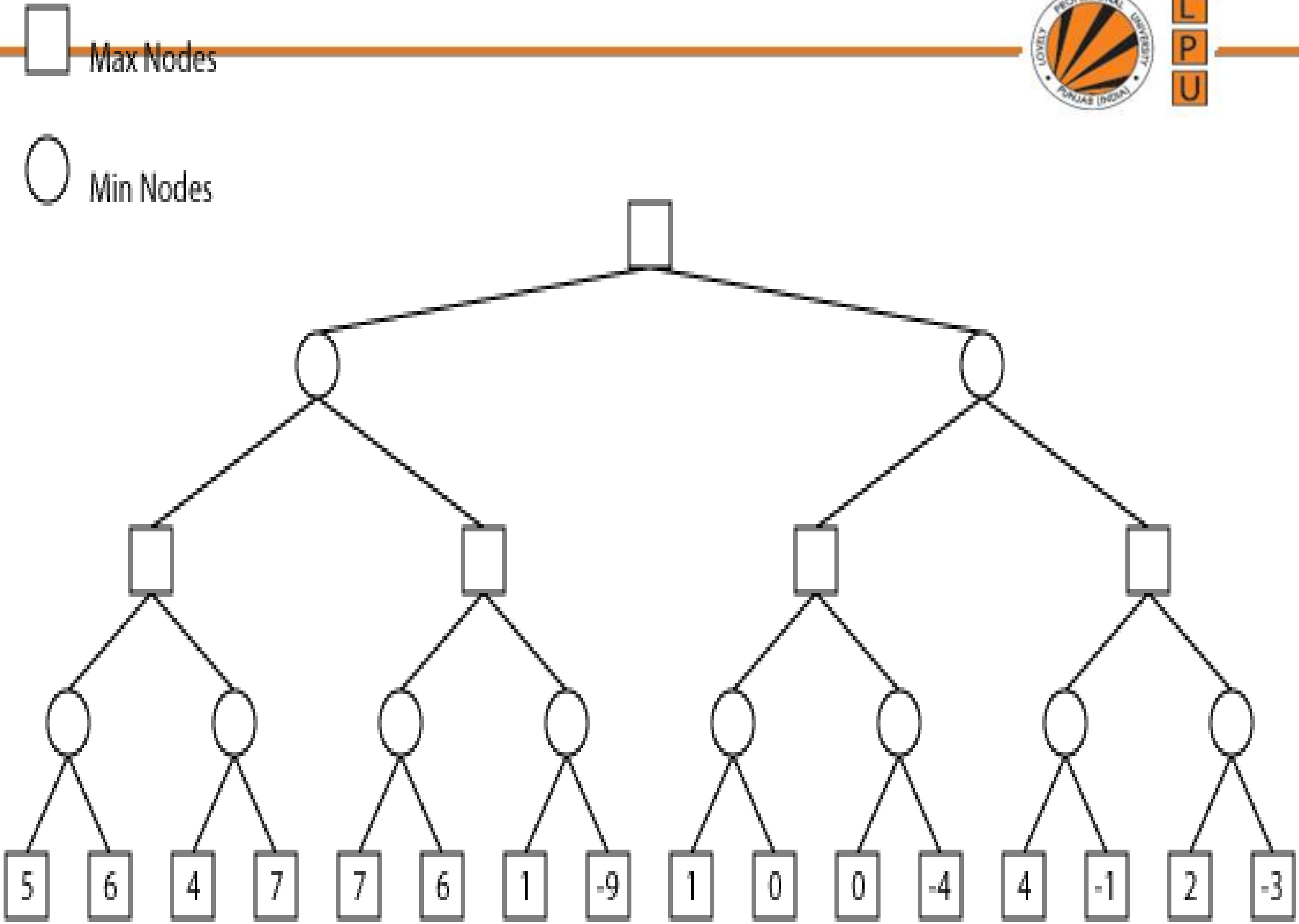


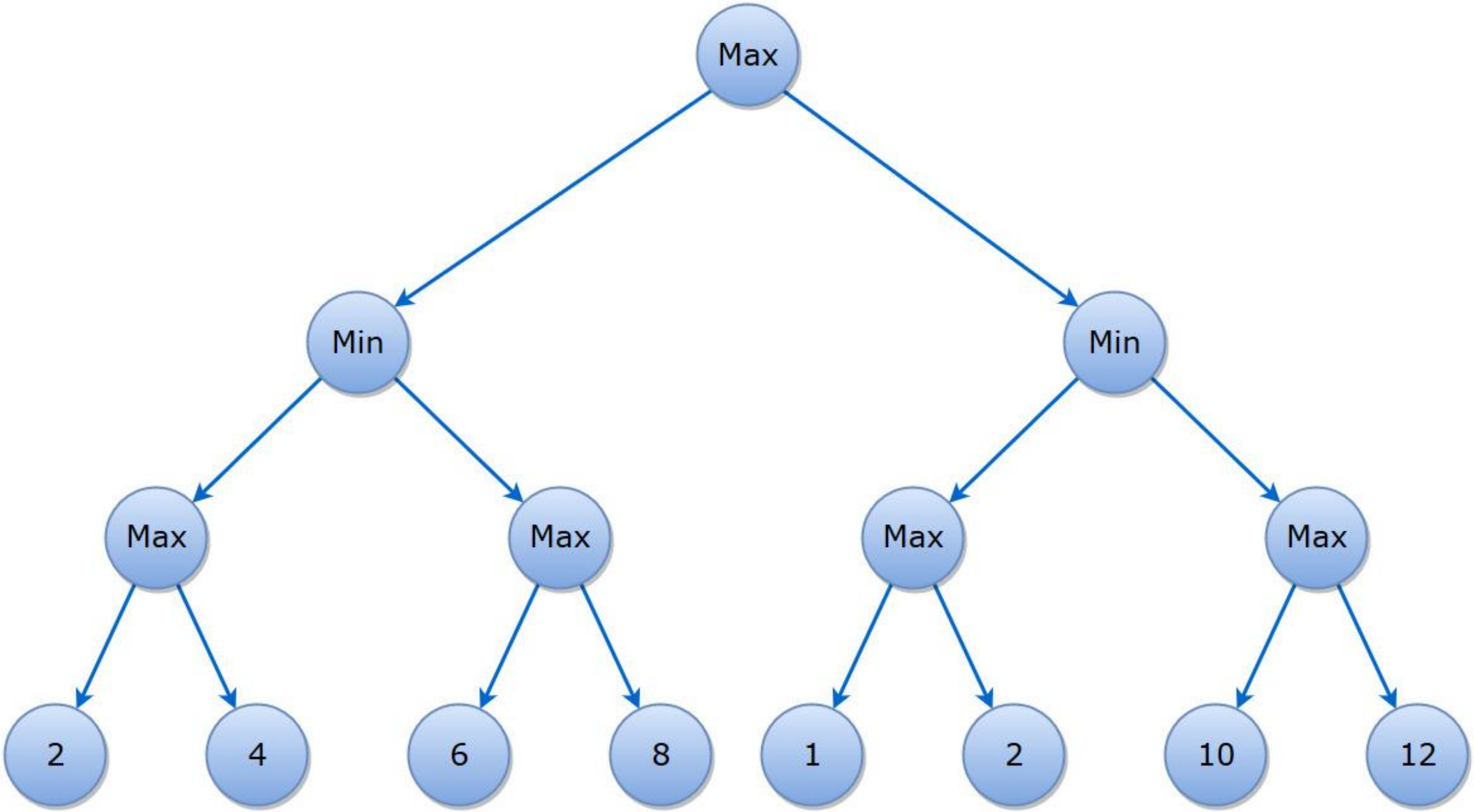
Alpha beta pruning. Example

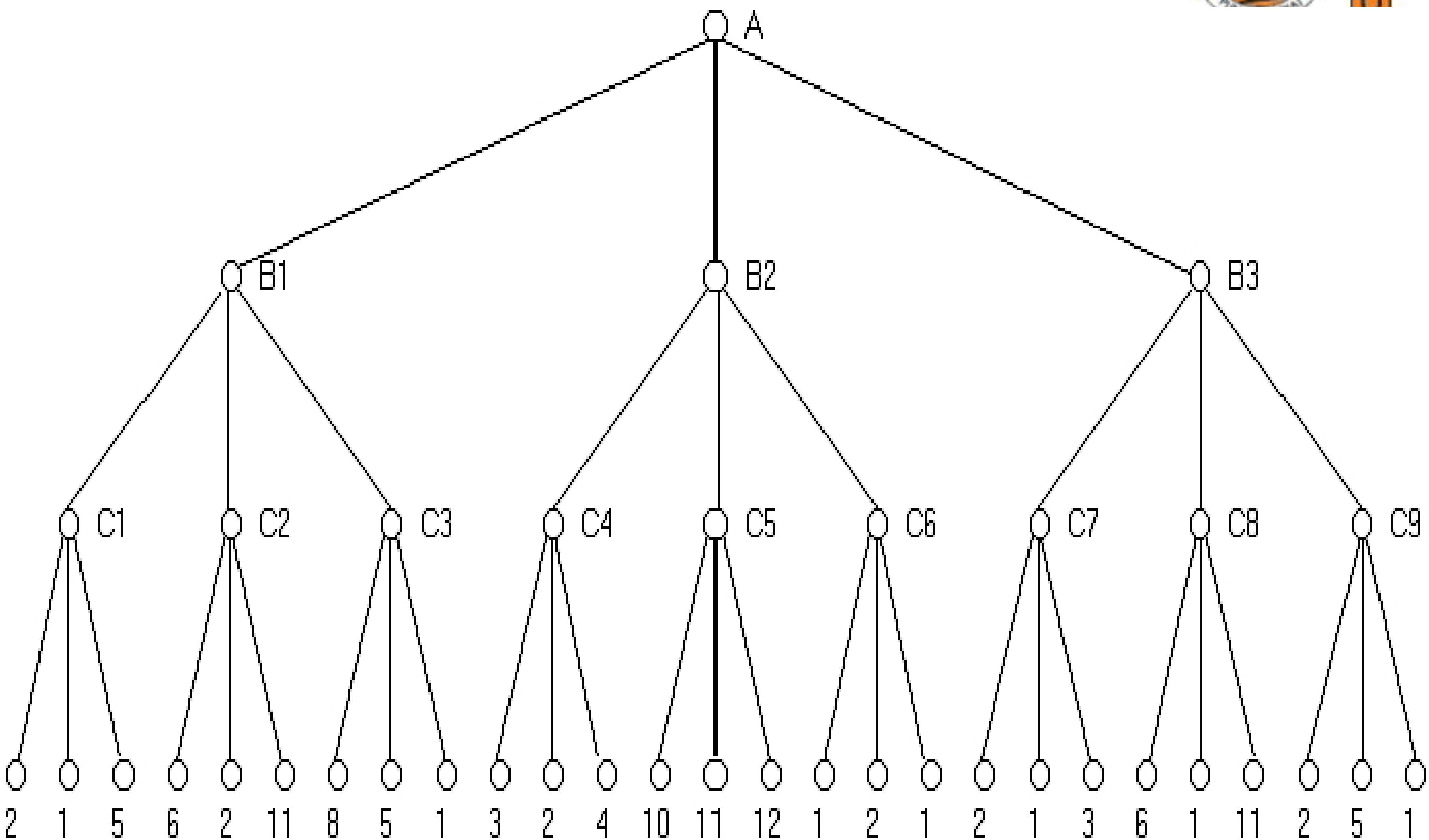


Alpha beta pruning. Example

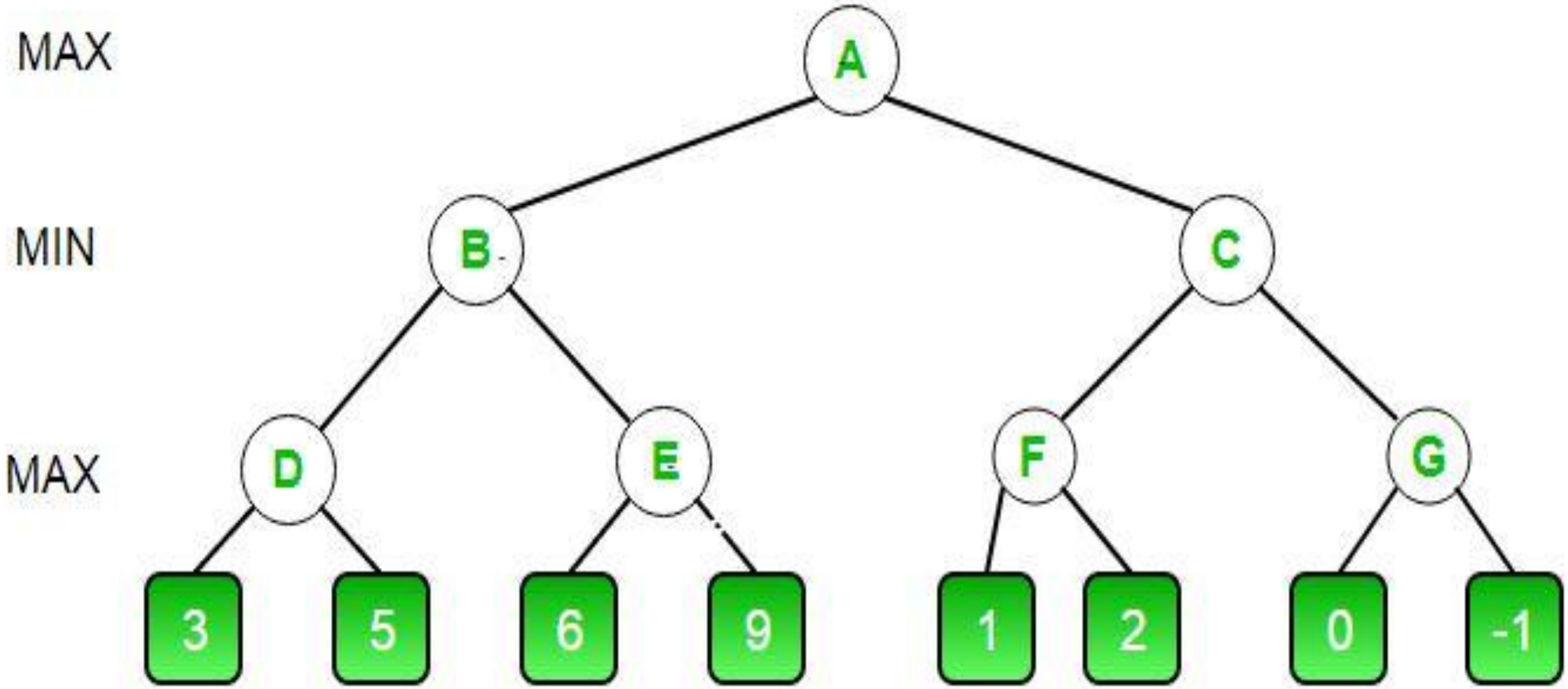


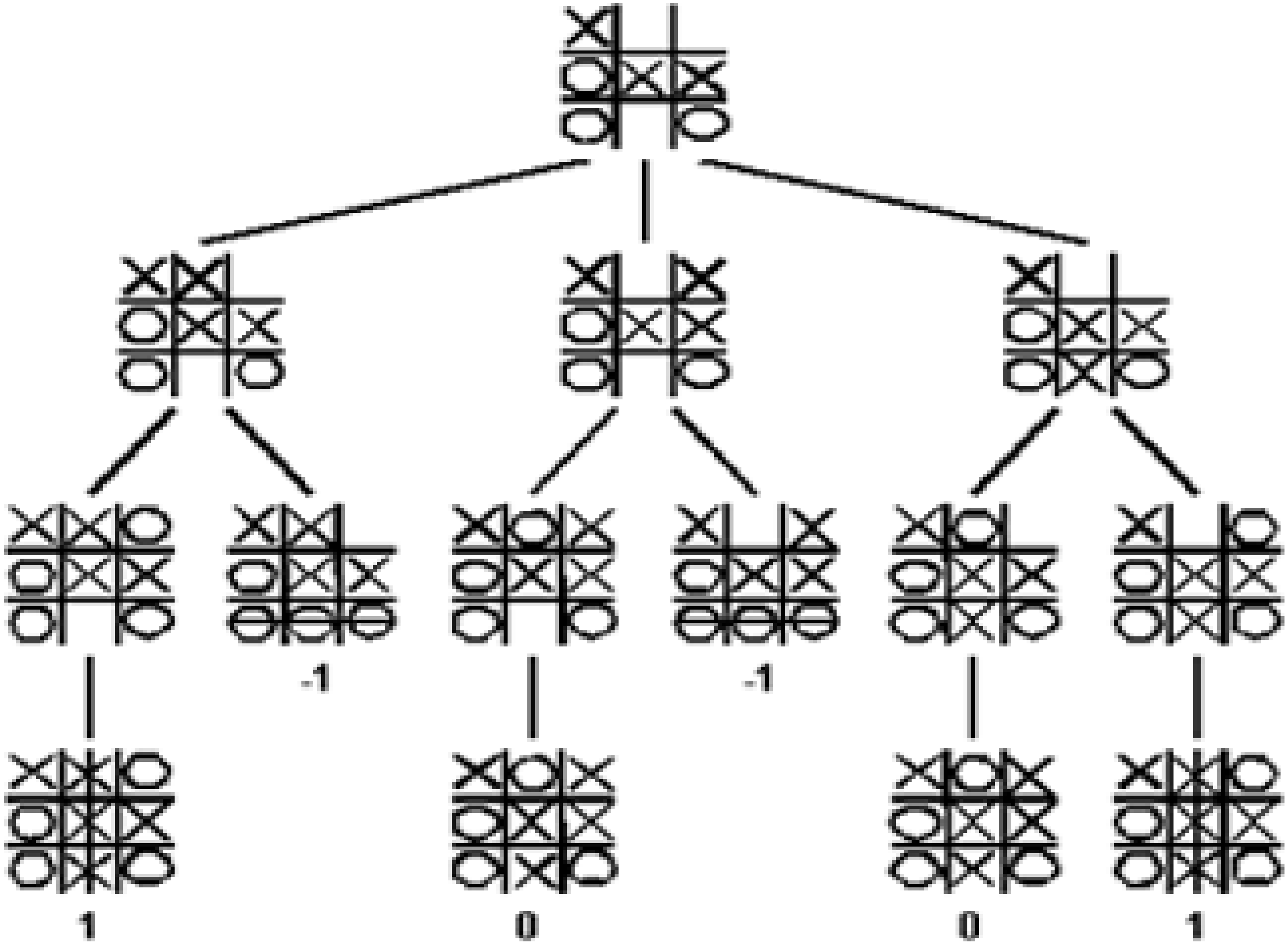

Max Nodes
Min Nodes
5
6
4
7
7
6
1
-9
1
0
0
-4
4
-1
2
-3

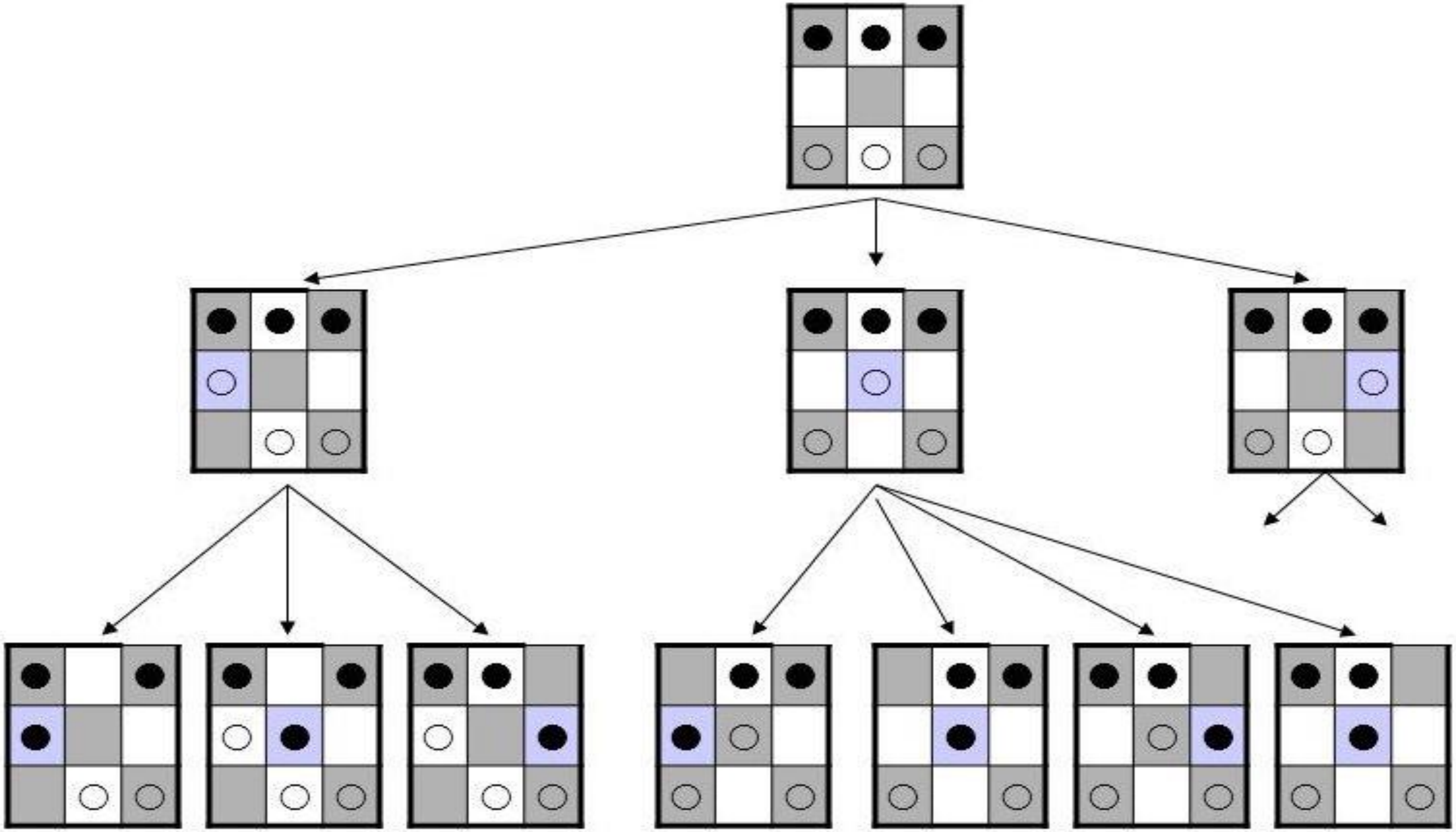

Max
Min
Min
Max
Max
Max
Max
2
4
6
8
1
2
10
12



A
B1
B2
B3
C1
C2
C3
C4
C5
C6
C7
C8
C9
2 1 5 6 2 11 8 5 1 3 2 4 10 11 12 1 2 1 2 1 3 6 1 11 2 5 1


MAX
MIN
MAX
A
B
C
D
E
F
G
3
5
6
9
1
2
0
-1


-1
-1
1
0
0
1



Thank You !!!