ARTIFICIAL INTELLIGENCE

AI-302 T

UNIT-III
Game Playing, Expert System

- Game Formulation
- Game Tree
- Adversarial Search
- Minimax Algorithm
- Alpha-beta Pruning
- Expert Systems

Why Study Game Playing?

- Games allow us to experiment with easier versions of real-world situations
- Hostile agents act against our goals
- Games have a finite set of moves
- Games are fairly easy to represent
- Good idea to decide about what to think
- Perfection is unrealistic, must settle for good
- One of the earliest areas of AI
  - Claude Shannon and Alan Turing wrote chess programs in 1950s
- The opponent introduces uncertainty
- The environment may contain uncertainty (backgammon)
- Search space too hard to consider exhaustively
  - Chess has about 10^{40} legal positions
  - Efficient and effective search strategies even more critical
- Games are fun to target!

Assumptions

- Static or dynamic?
- Fully or partially observable?
- Discrete or continuous?
- Deterministic or stochastic?
- Episodic or sequential?
- Single agent or multiple agent?

MAEs and Games

- Multi-agent environment: every agent needs to consider the actions of the other agents, in order to optimize its own welfare
  - Normally considered in terms of economies
  - Cooperative: Agents act collectively to achieve a common goal
  - Competitive:
    - Agents compete against each other
    - Their goals are in conflict
    - Gives rise to the concept of adversarial search problems – often known as games.

Game Theory

- It's a branch of economics
  - Views any MAE as a game provided that the impact of each agent on the other is "significant", i.e., able to affect the actions of the other agent(s)
- In AI, "game" is a specialized concept:
  - Deterministic, fully-observable environments
  - Two agents whose actions must alternate
  - Utility values at the end of the game are always equal and opposite
    - +1 = Chess winner
    - -1 = Chess looser.

AI Games

- Tackled by Konrad Zuse, Claude Shannon, Norbert Wiener, Alan Turing
  - Have seen lot of successes recently, e.g., DeepBlue
- Game states are easy to represent:
- Agents restricted by a limited action rules
- Outcomes defined by precise rules
- Games: interesting *because* they are hard to solve:
  - Chess: average branching factor of 35
  - If 50 moves by each player, search tree has 35^{100} nodes!

Zero-Sum Games

- Focus primarily on “adversarial games”
- Two-player, zero-sum games

As Player 1 gains strength →
Player 2 loses

← strength and vice

versa

The sum of the two strengths is constant, 0 in many

Search Applied to Adversarial Games

- Initial state
  - Current board position (description of current game state)
- Operators
  - Legal moves a player can make
- Terminal nodes
  - Leaf nodes in the tree
  - Indicate the game is over
- Utility function
  - Payoff function
  - Value of the outcome of a game
  - Example: tic tac toe, utility is -1, 0, or 1

Game Formulation

- S0:The **initial state**, 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**, 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 or terminal nodes**.
- UTILI TY(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 1/2

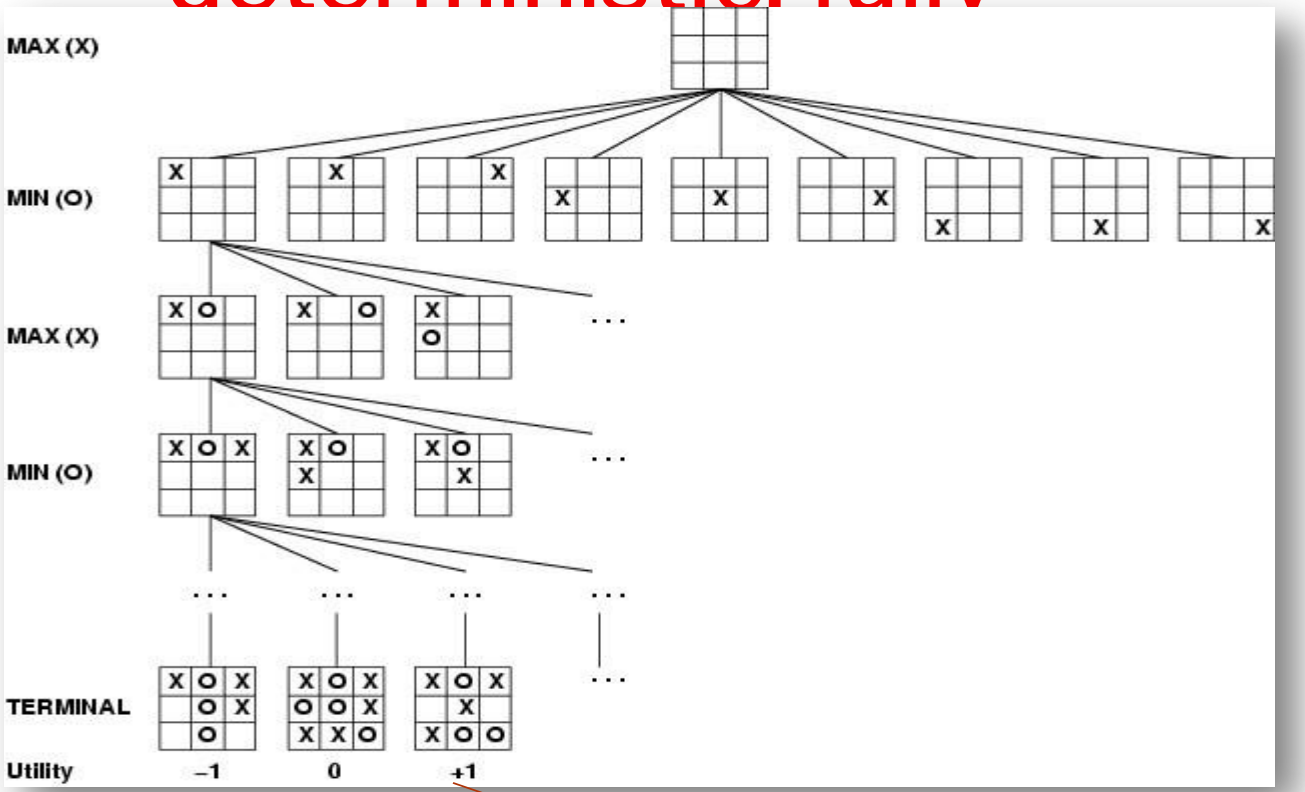
Using Search

- Search could be used to find a perfect sequence of moves except the following problems arise:
  - There exists an adversary who is trying to minimize your chance of winning every other move
    - You cannot control his/her move
  - Search trees can be very large, but you have finite time to move
    - Chess has 10^{40} nodes in search space
    - With single-agent search, can afford to wait
    - Some two-player games have time limits
    - Solution?
      - Search to n levels in the tree (n **ply** for each player)
      - Evaluate the nodes at the nth level
      - Head for the best looking node

Games vs. Search problems

- In typical search problems, we optimize a measure to acquire the goal: there is no opponent
- In games, there is an "Unpredictable" opponent
  - Need to specify a move for every possible opponent reply
  - Strict penalty on an inefficient move
  - Stringent time constraints
  - Unlikely to find goal, must approximate
  - Requires some type of a decision to move the search forward.

Game tree (2-player, deterministic, fully



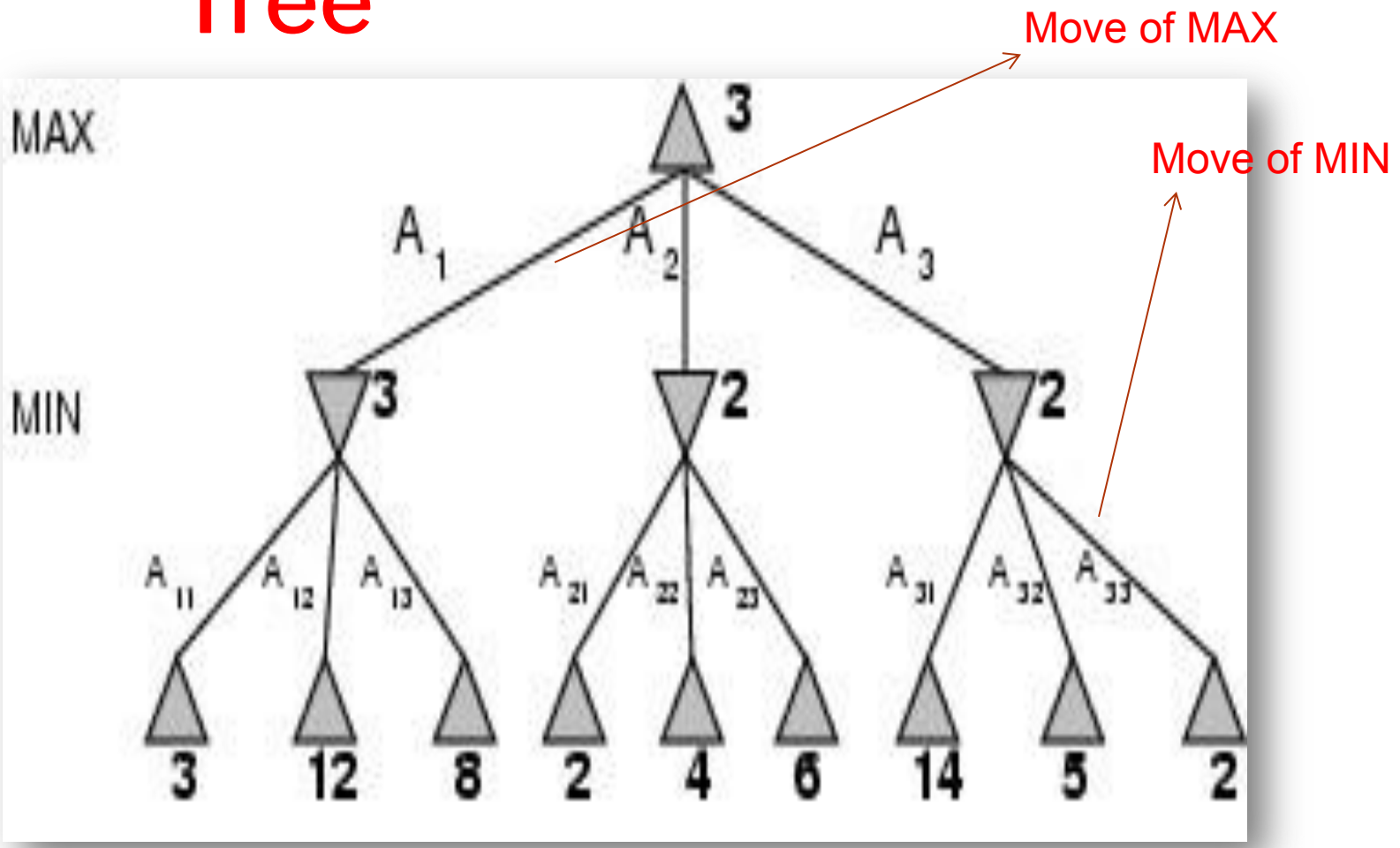
This terminal state is one of the worst for MAX and one of the best for MIN

This terminal state is one of the best for

Optimal Strategy For MAX

- This is a MAE
  - In discovering the sequence of optimal moves by MAX, we have to consider that MIN is taking moves as well
  - So, the optimal strategy will tell how should MAX play against MIN (by considering the moves of the latter), in order to win the game
- Let us consider a shortened game tree of TIC-TAC-TOE
  - Numbers are the utilities
  - Ply: a single move by MAX or MIN.

Shortened Game Tree



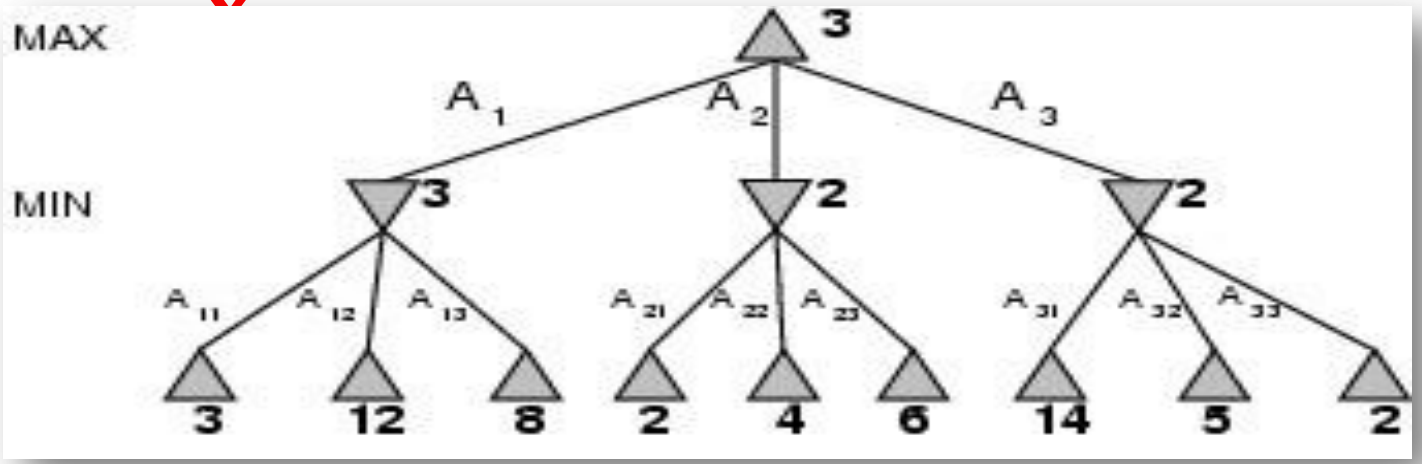
Minimax

- When it is the turn of MAX, it will always take an action in order to maximize its utility, because it's winning configurations have high utilities
- When it is the turn of MIN, it will always take an action in order to minimize its utility, because it's winning configurations have low utilities
- In order to implement this, we need to define a measure in each state that takes the move of the opponent into account:
  - This measure is called Minimax.

Minimax

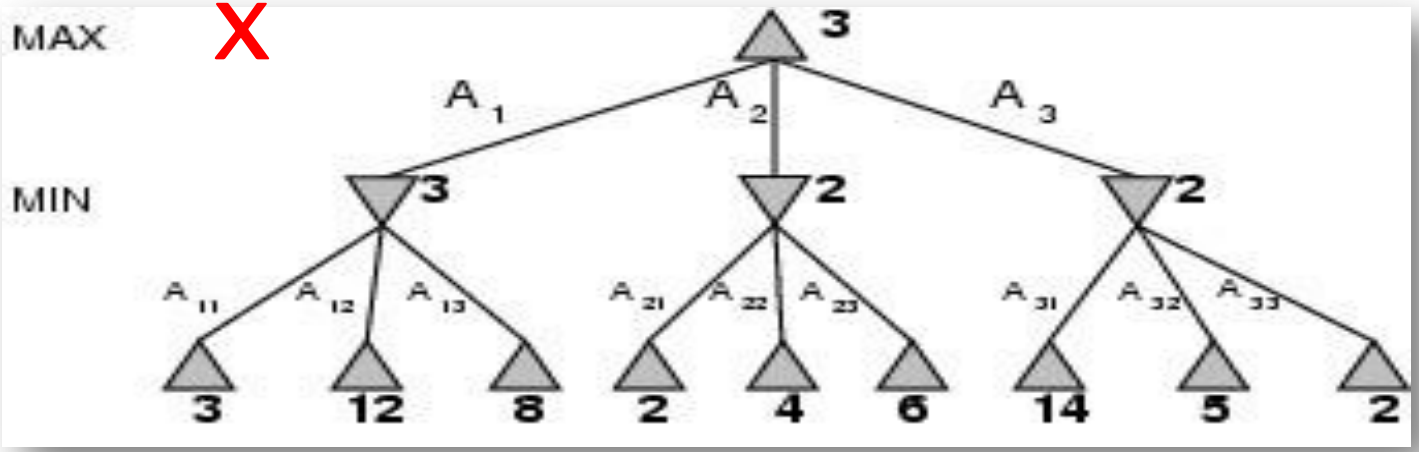
- Minimax represents the utility of a state, *given that both MAX and MIN will play optimally till the end of the game*
- In any state *s*, one or more actions are possible
- For every possible new state that can be transited into from *s*, we compute the minimax value
- The term "Minimax" is used because:
  - the opponent is always trying to minimize the utility of the player, and
  - the player is always trying to maximize this minimized selection of the opponent.
- Confused? See next slide…

Minima
x



- Consider "3" at Level 1: MIN selects an action (A11) that leads to a state of minimum utility for MAX, i.e., minimum{3,12,8}
- Consider "3" at Level 0: MAX selects an action (A1) that leads to a state of maximum utility for MIN, i.e., maximum{3,2,2}
- Both are opposing what is best for the other.

Minima x



- At each node, MAX will always select the action with highest minimax value (it wants to reach states with higher utilities)
- At each node, MIN will always select the action with lowest minimax value (it wants to reach states with lower utilities).

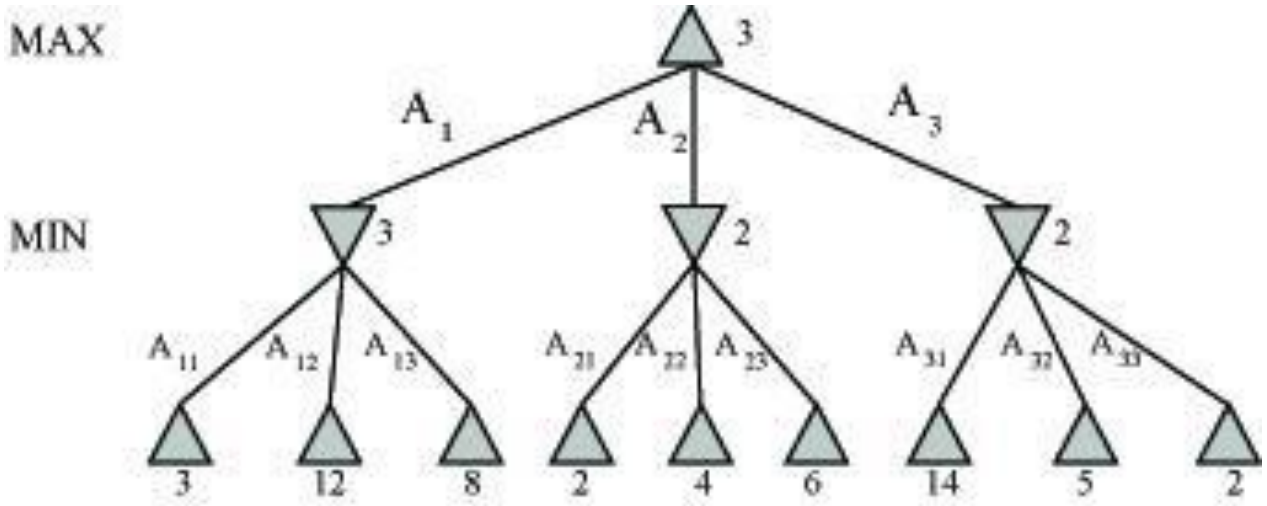
Minimax

- Mathematically,

$$\text{MINIMAX}(s) = \begin{cases} \text{UTILITY}(s) & \text{if TERMINAL-TEST}(s) \\ \max_{a \in Actions(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MAX} \\ \min_{a \in Actions(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MIN} \end{cases}$$

Minimax Algorithm

- Search the tree to the end
- Assign utility values to terminal nodes
- Find the best move for MAX (on MAX's turn), assuming:
  - MAX will make the move that maximizes MAX's utility
  - MIN will make the move that minimizes MAX's utility
- Here, MAX should make the leftmost move



Minimax

```
function MINIMAX-DECISION(state) returns an action
    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 ← −∞
    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 ← ∞
    for a, s in SUCCESSORS(state) do
        v ← MIN(v, MAX-VALUE(s))
    return v
```

Depth-First Exploration

Recursion: Winds all the way to the terminal nodes, and then unwinds back by backing up the values

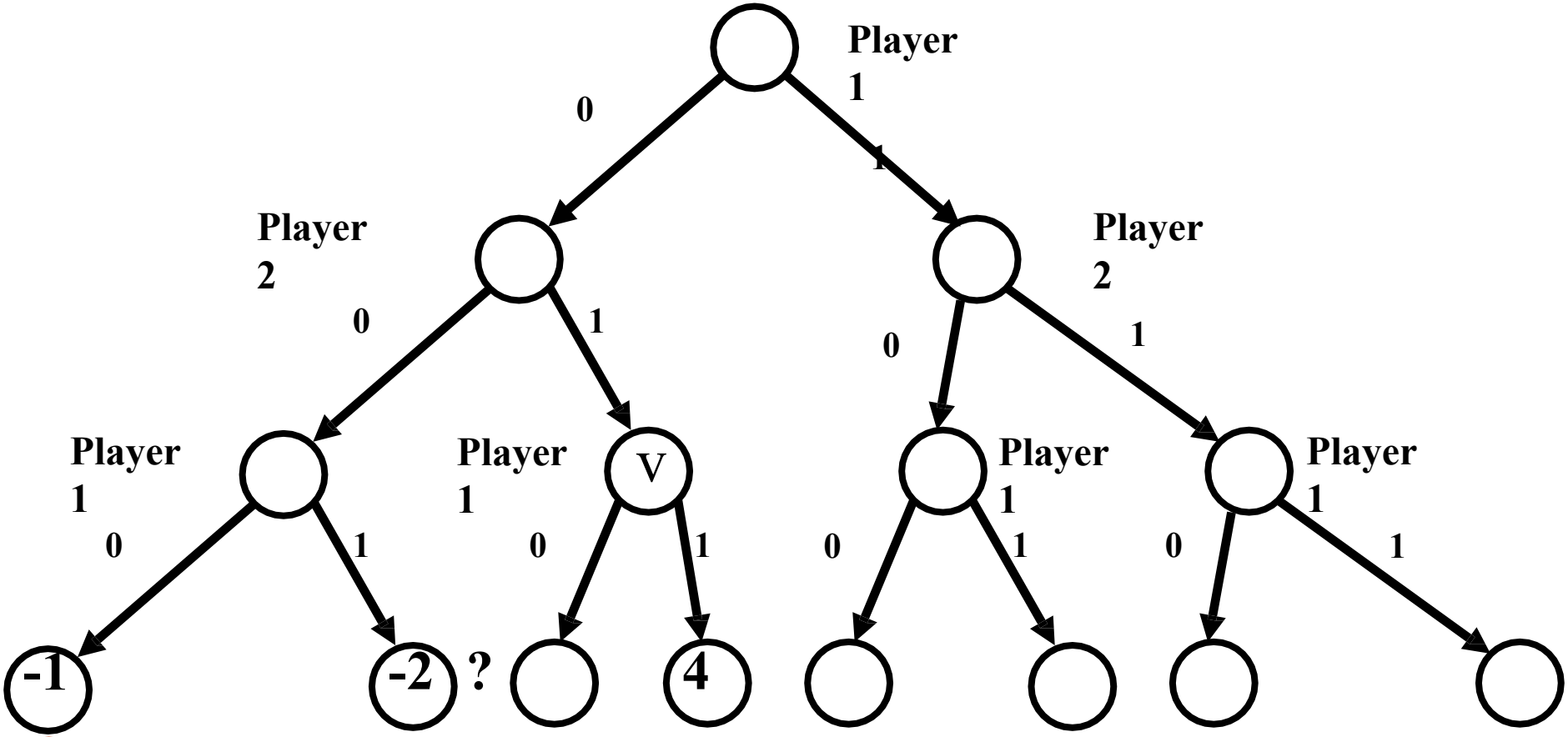
Min(minimax) for MIN moves
Max(minimax) for MAX moves

Properties of Minimax

- Complete?Yes (if tree is finite)
- Optimal?Yes (against an optimal opponent)
- Time complexity? $O(b^m)$
- Space complexity? O(bm) (depth-first exploration)
- For chess, b ≈ 35, m ≈100 for "reasonable" games exact solution completely infeasible
  - We need to think of a way to cut down the number of search paths.

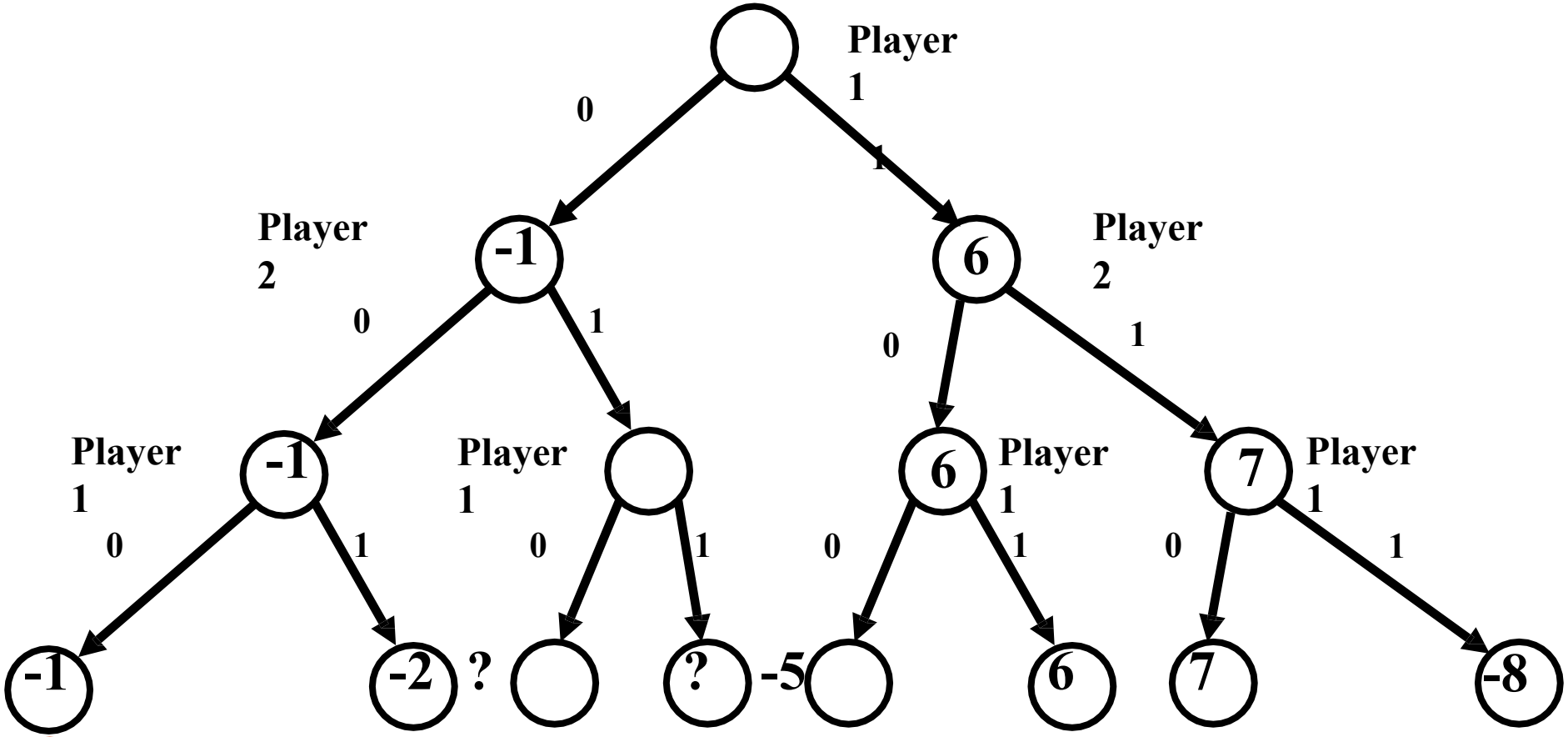
Do we need to see all the leaves?

- Do we need to see the values of the question marks here?



Do we need to see all the leaves?

- Do we need to see the values of the question marks here?



Alpha – Beta Pruning

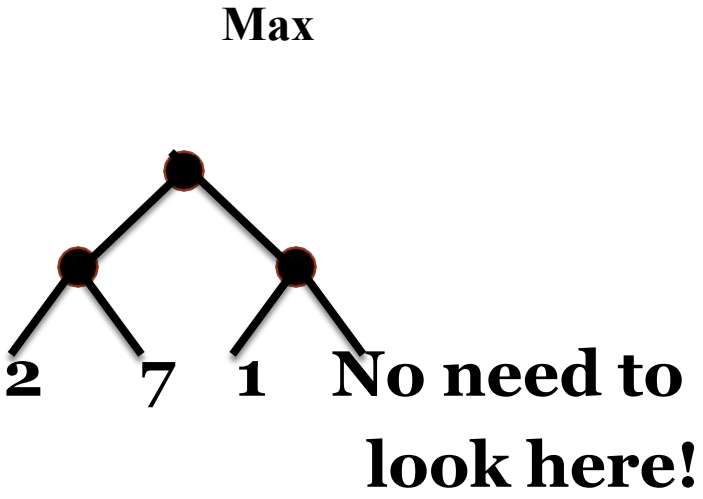
- **Problem with Minimax Search:**
  - Number of states to examine is exponential in the depth of tree.
- **Bad News:**
  - We can't eliminate the exponent.
- **Good News:**
  - We can cut by half the number of nodes to examine
- **Trick:** To compute the correct minimax decision without looking at every node in the game tree.
  - Prunes away branches that can not possibly influence decision.

Alpha-beta pruning

- Pruning = cutting off parts of the search tree (because you realize you don't need to look at them)
  - When we considered A* we also pruned large parts of the search tree
- Maintain alpha = value of the best option for player-1 (Max) along the path so far
- Beta = value of the best option for player-2 (Min) along the path so far

Alpha-Beta Pruning

- Typically, can only look 3-4 ply in allowable chess time
- Alpha-beta pruning simplifies search space without eliminating optimality
  - By applying common sense
  - If one route allows queen to be captured and a better move is available
  - Then don't search further down bad path
  - If one route would be bad for opponent, ignore that route also

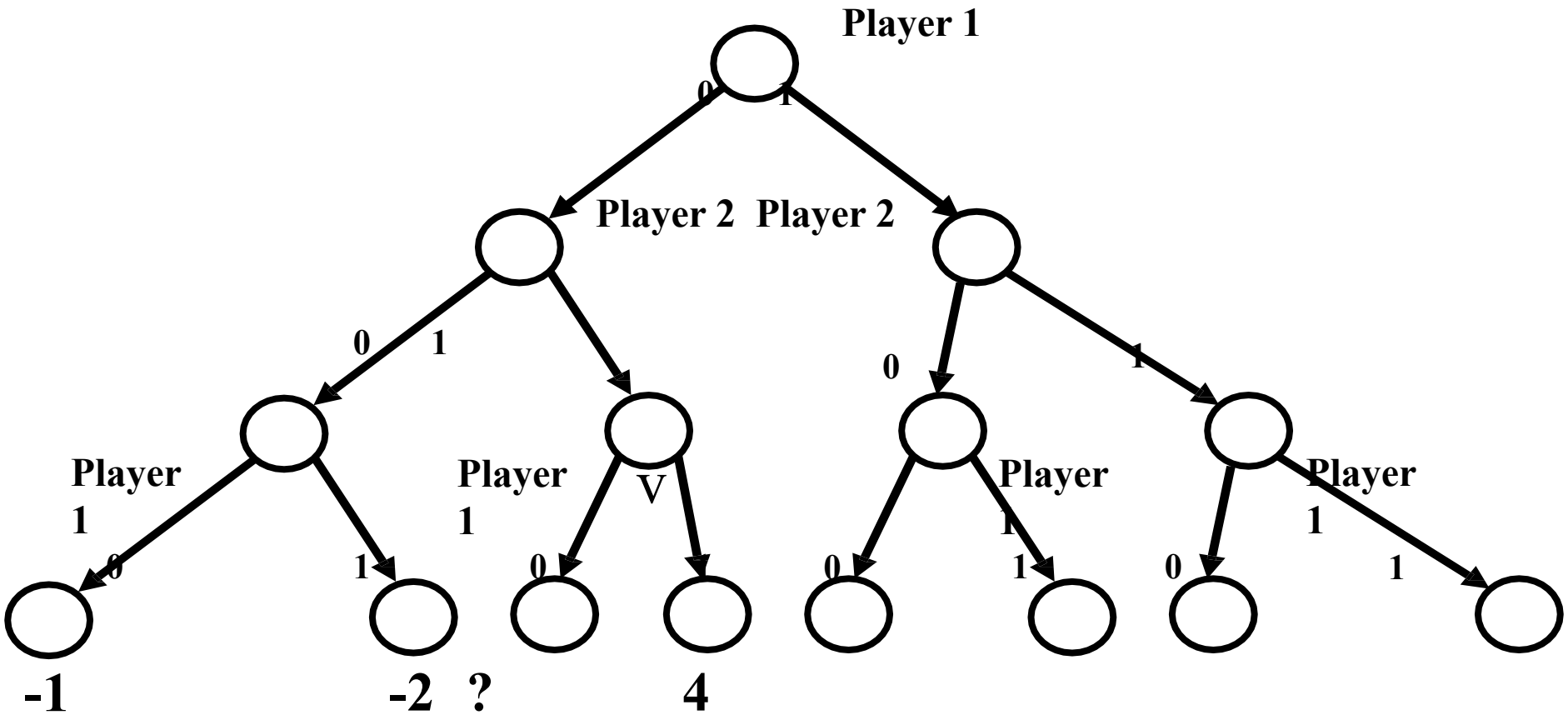


Maintain [alpha, beta] window at each node during depth-first search

alpha = lower bound, change at max levels
beta = upper bound, change at min levels

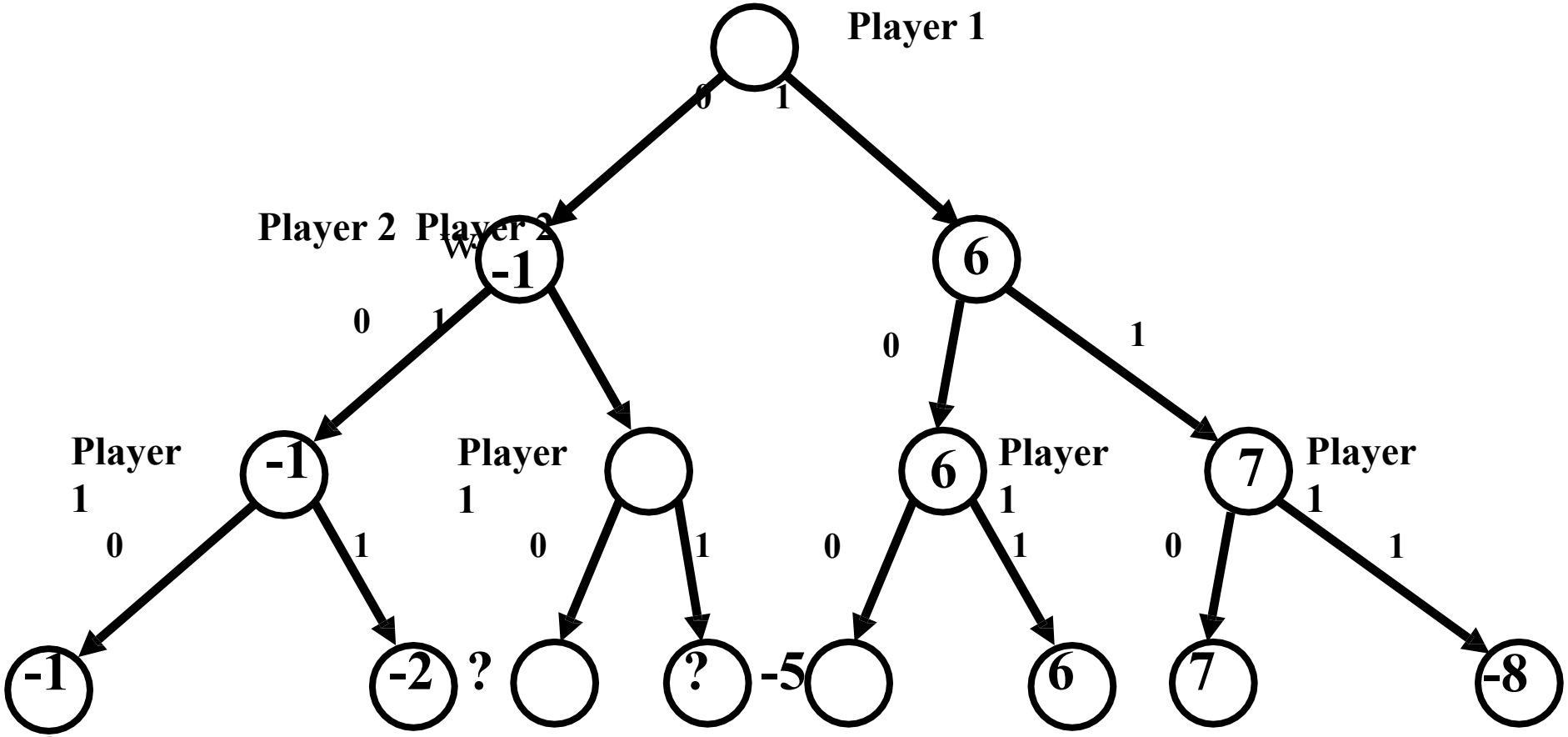
Pruning on Beta

- Beta at node v is -1
- Since value of node v is going to be at least 4, -1 route will be preferred
- No need to explore this node further

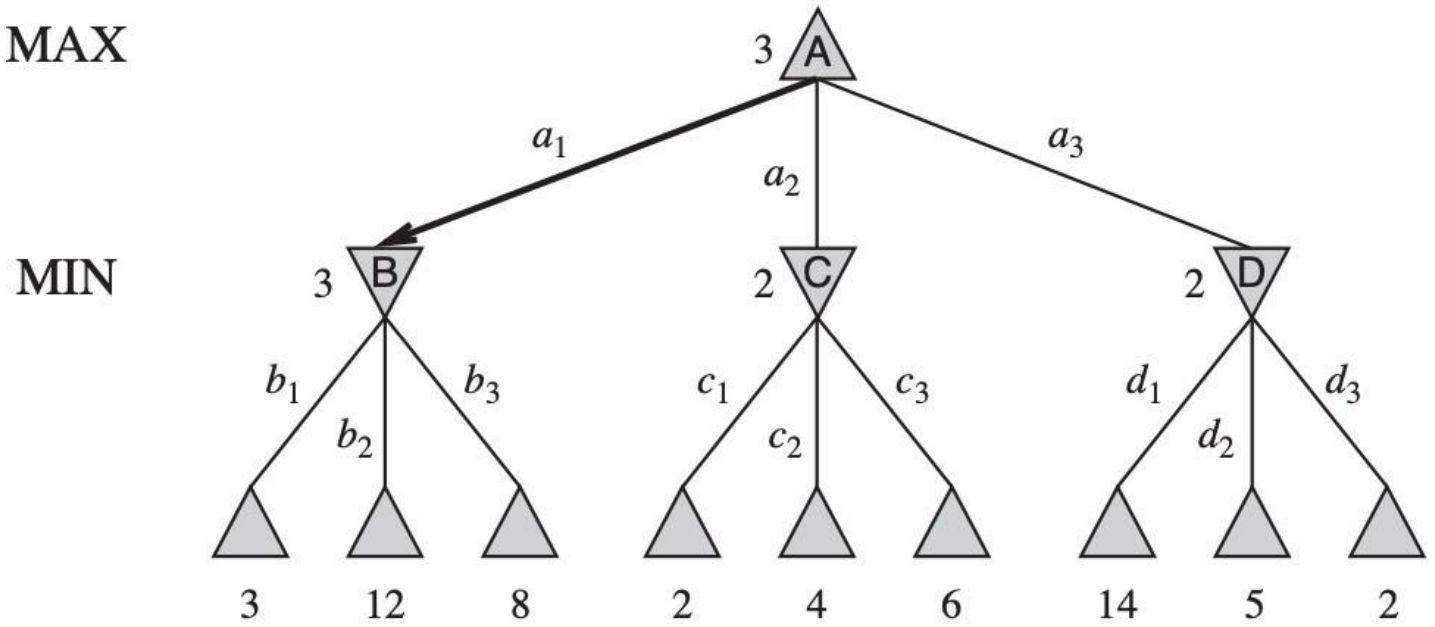


Pruning on alpha

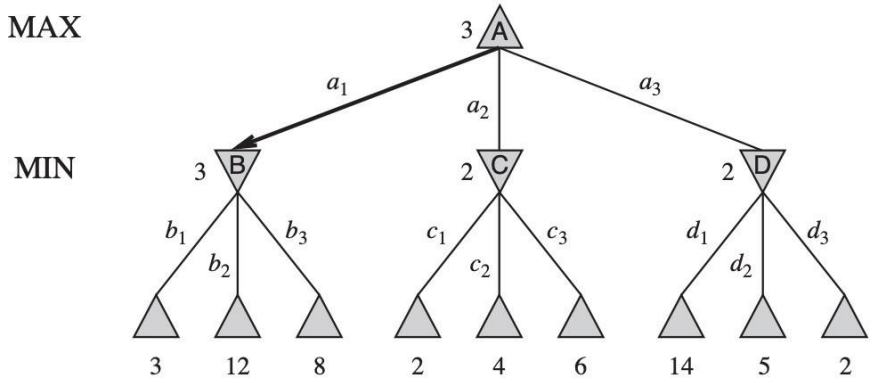
- Alpha at node w is 6
- Value of node w is going to be at most -1, so the 6 route will be preferred
- No need to explore this node further

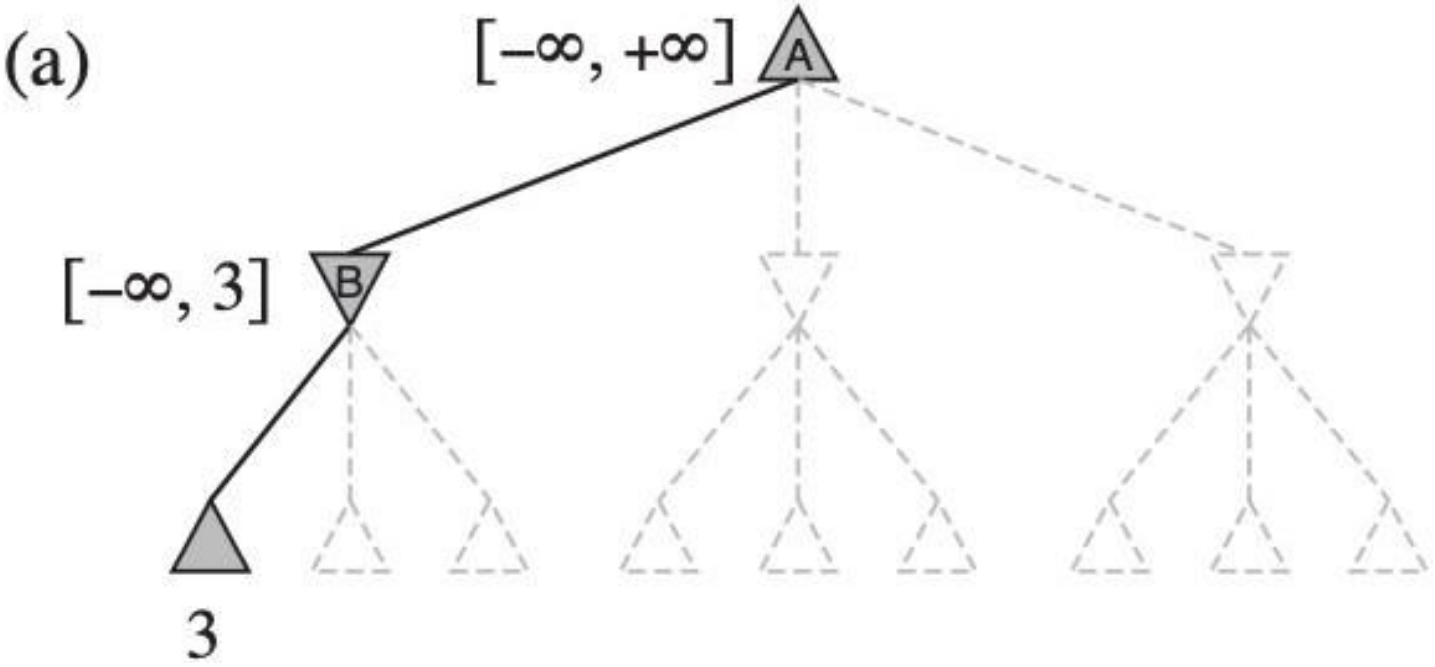


Alpha-beta pruning

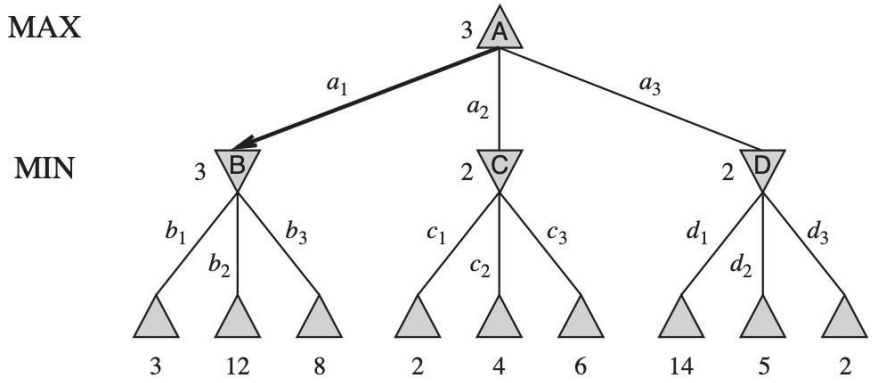


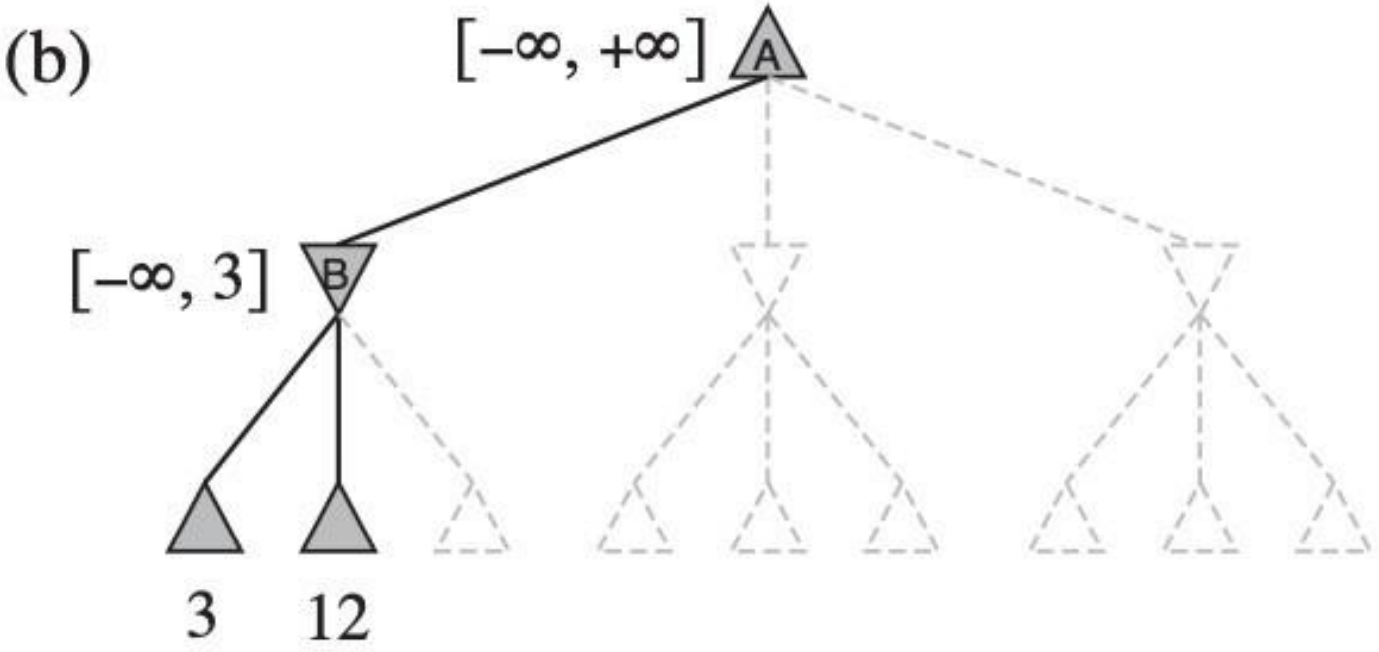
Alpha – Beta Pruning



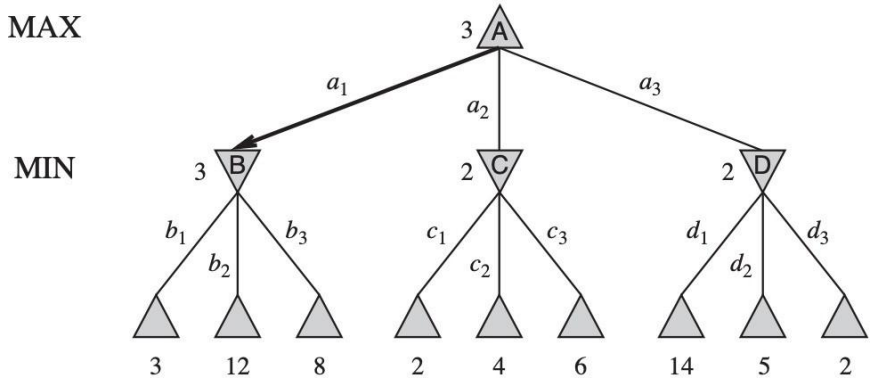


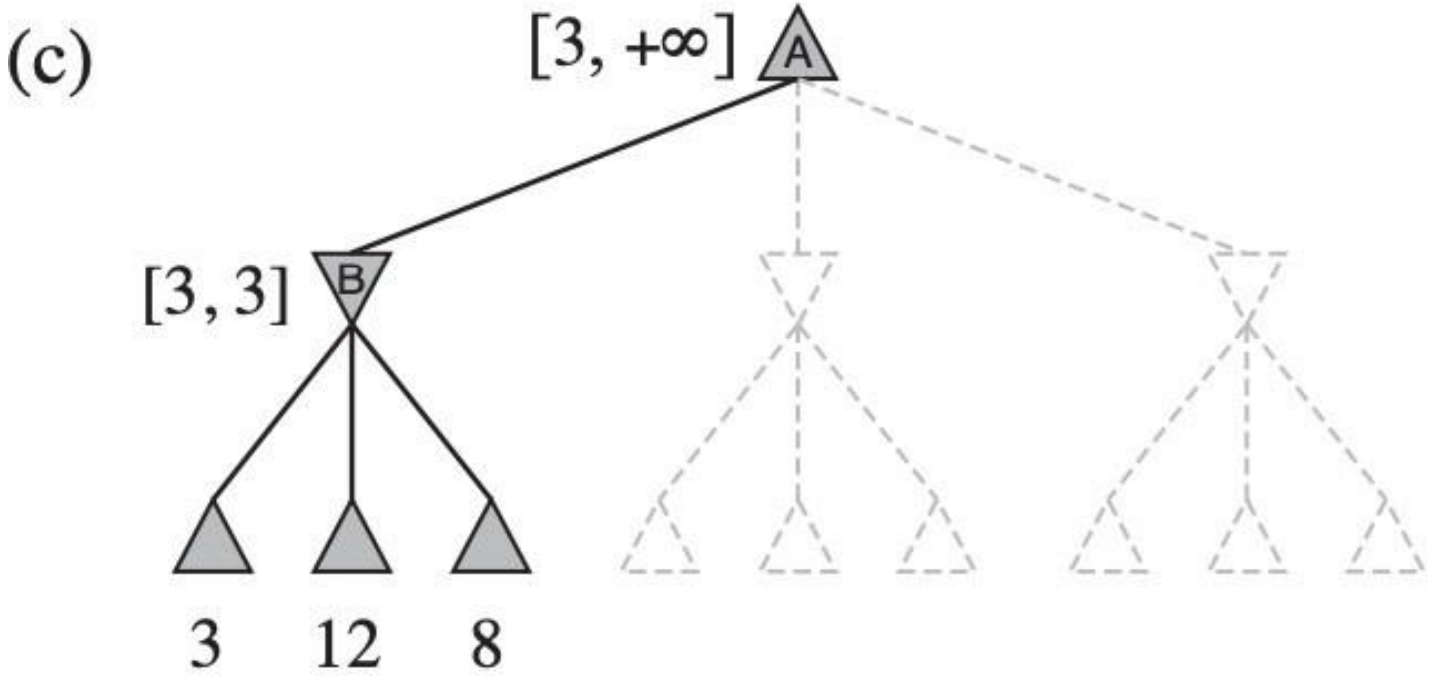
Alpha – Beta Pruning



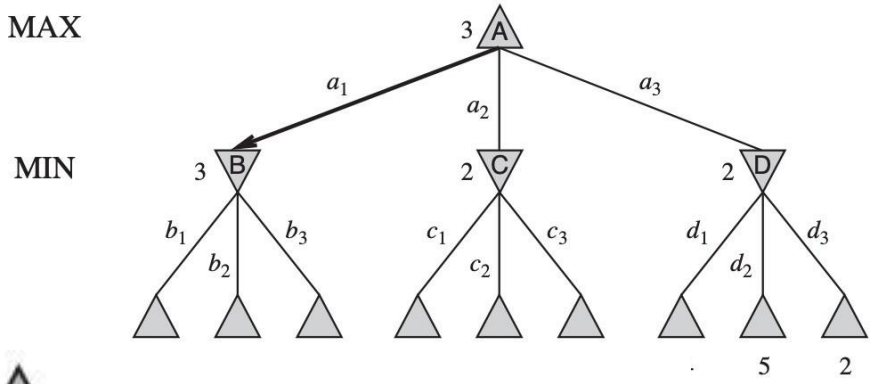


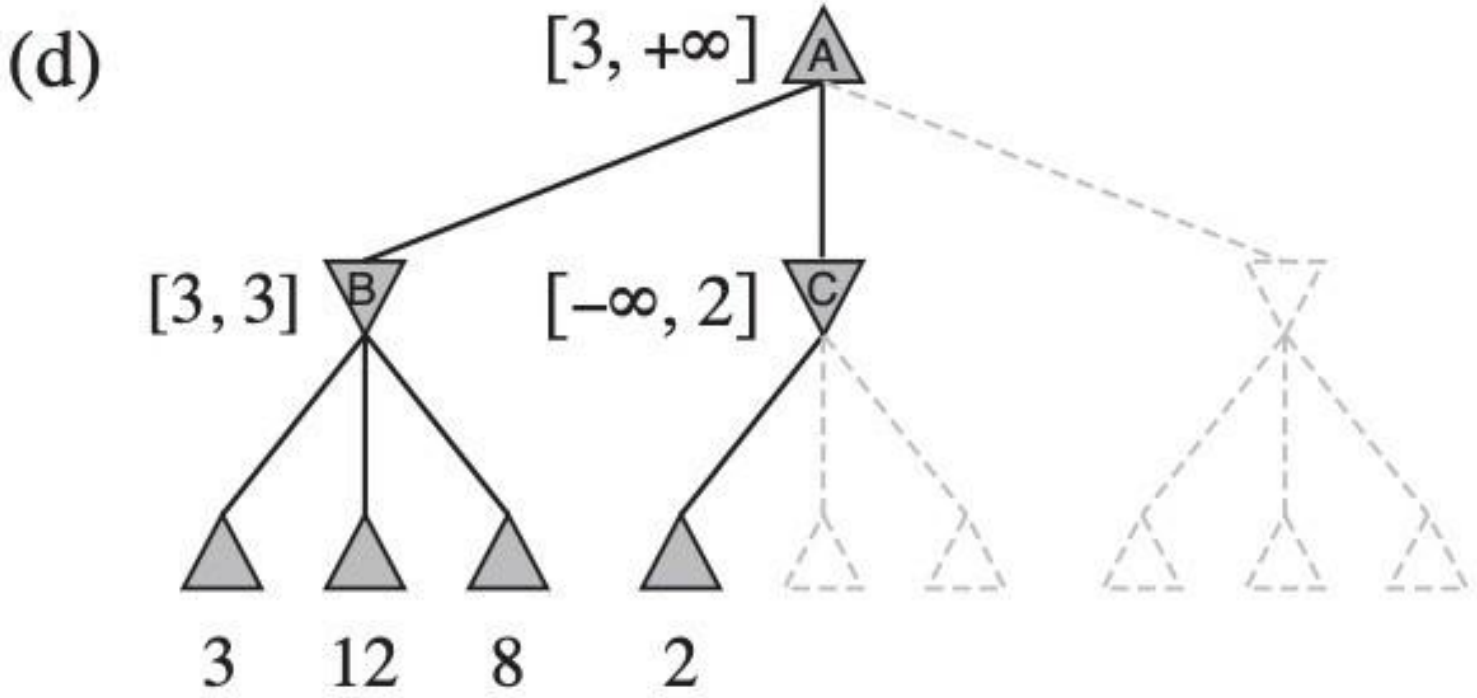
Alpha – Beta Pruning



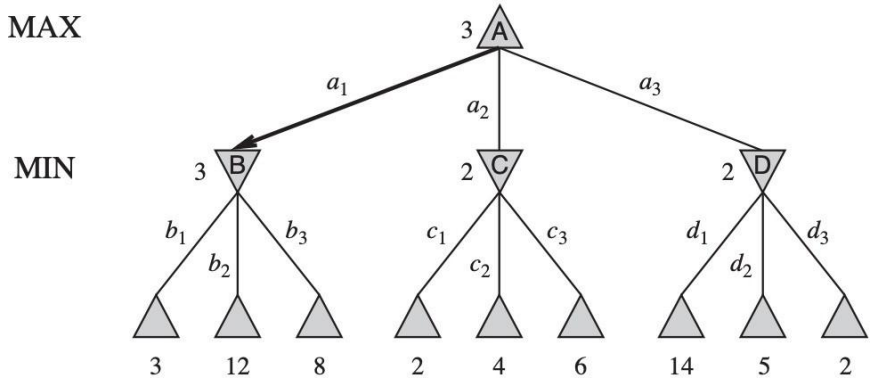


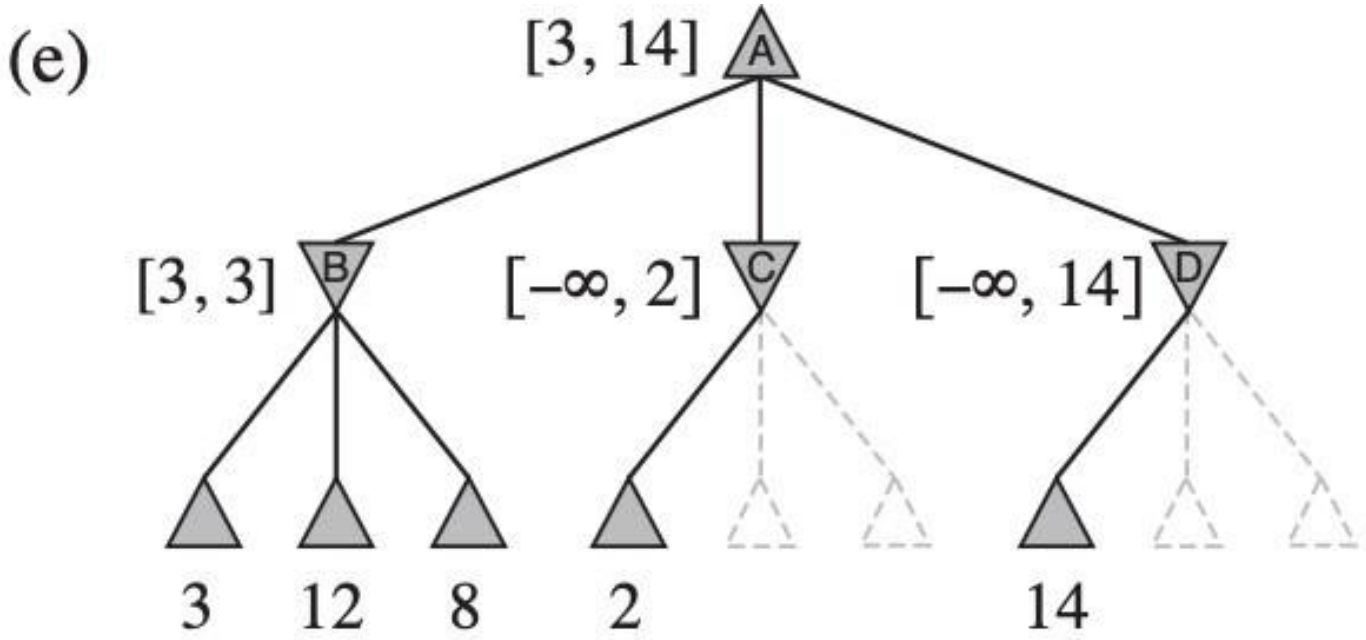
Alpha – Beta Pruning



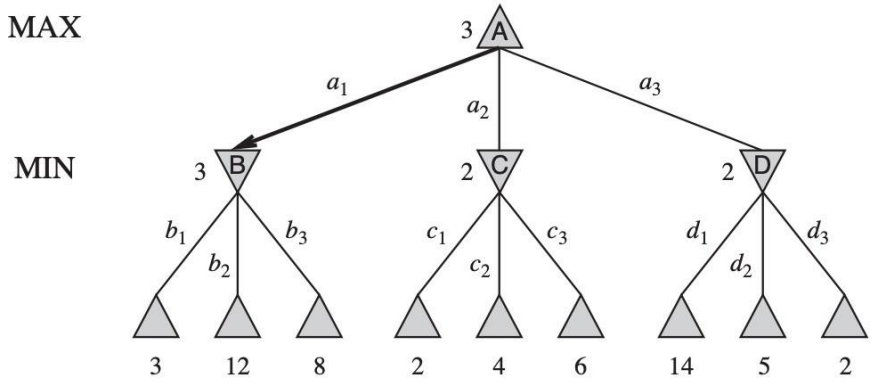


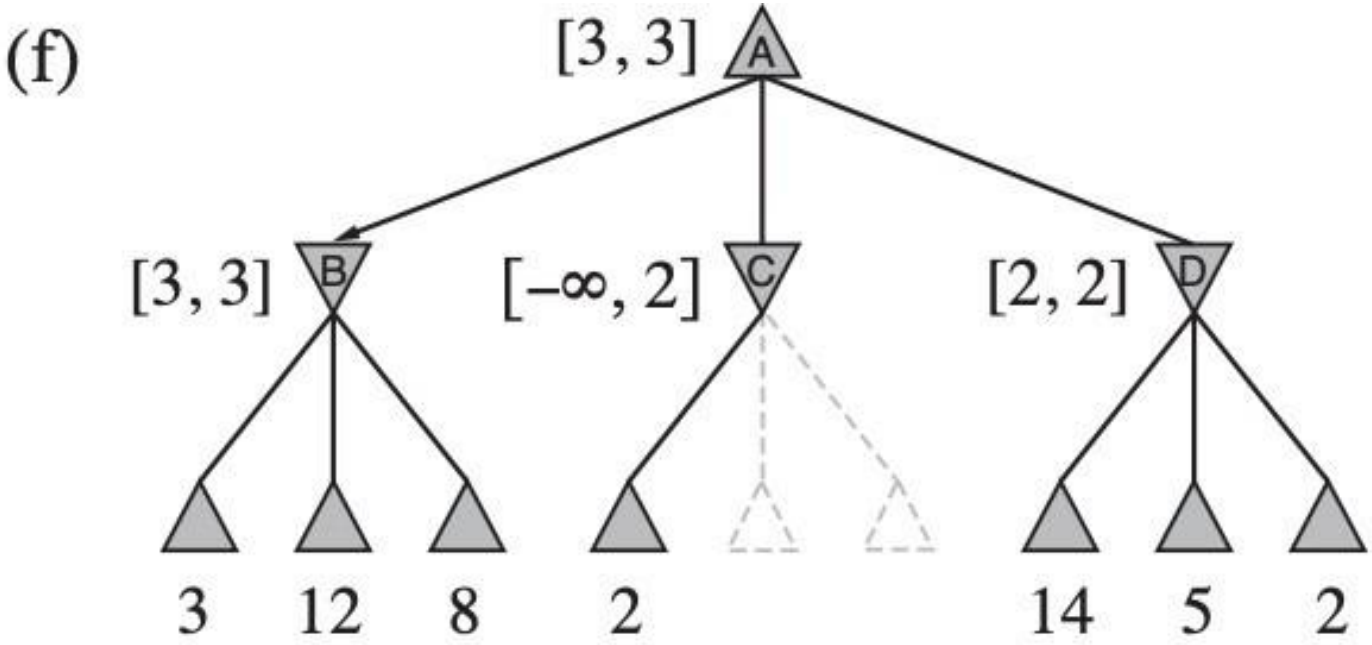
Alpha – Beta Pruning





Alpha – Beta Pruning



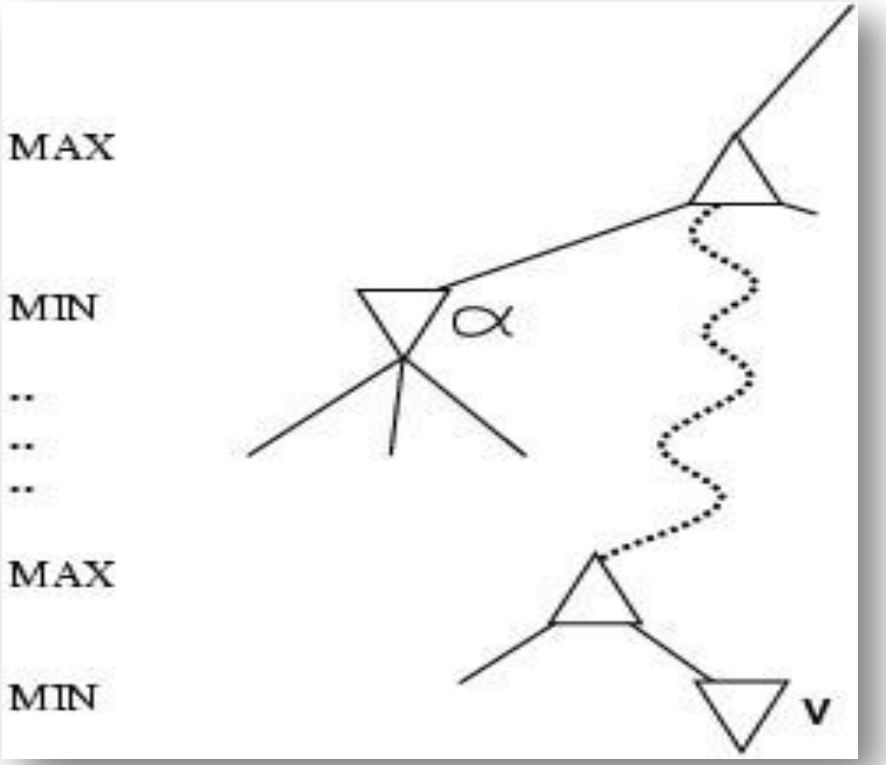


Benefits of alpha-beta pruning

- Without pruning, need to examine $O(b^m)$ nodes
- With pruning, depends on which nodes we consider first
- If we choose a random successor, need to examine $O(b^{3m/4})$ nodes
- If we manage to choose the best successor first, need to examine $O(b^{m/2})$ nodes
  - Practical heuristics for choosing next successor to consider get quite close to this
- Can effectively look twice as deep!
  - Difference between reasonable and expert play

Why is it called α-β?

- α is the value of the best (i.e., highest-value) choice found so far at any choice point along the path for *MAX*
- If *v* is worse than α, *MAX* will avoid it
  - prune that branch
- Define β similarly for *MIN.*
- β = the value of the best (i.e., lowest-value) choice we have found so far at any choice



The α-β algorithm

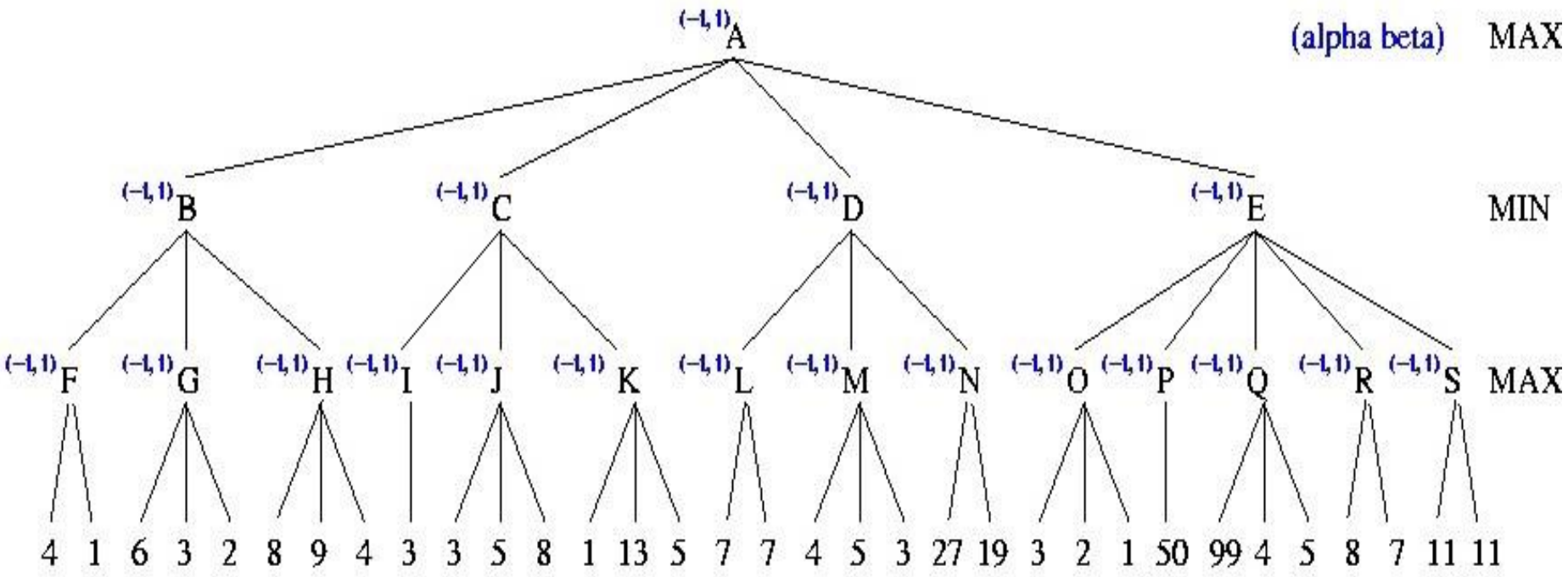
```
function ALPHA-BETA-SEARCH(state) returns an action
    inputs: state, current state in game
    v ← MAX-VALUE(state, −∞, +∞)
    return the action in SUCCESSORS(state) with value v

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
```

```
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
```

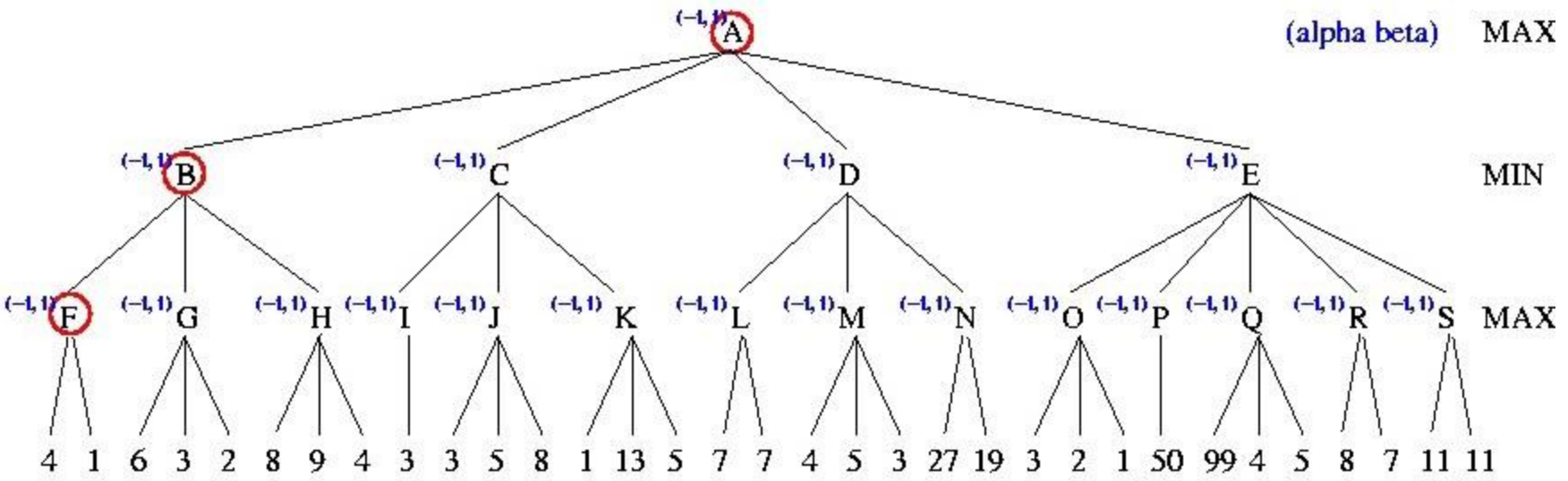
Properties of α-β

- Pruning does not affect the final result
- Good move ordering improves effectiveness of pruning
- With "perfect ordering," time complexity = $O(b^{m/2})$
  - doubles depth of search
- A simple example of the value of reasoning about which computations are relevant (a form of meta-reasoning)

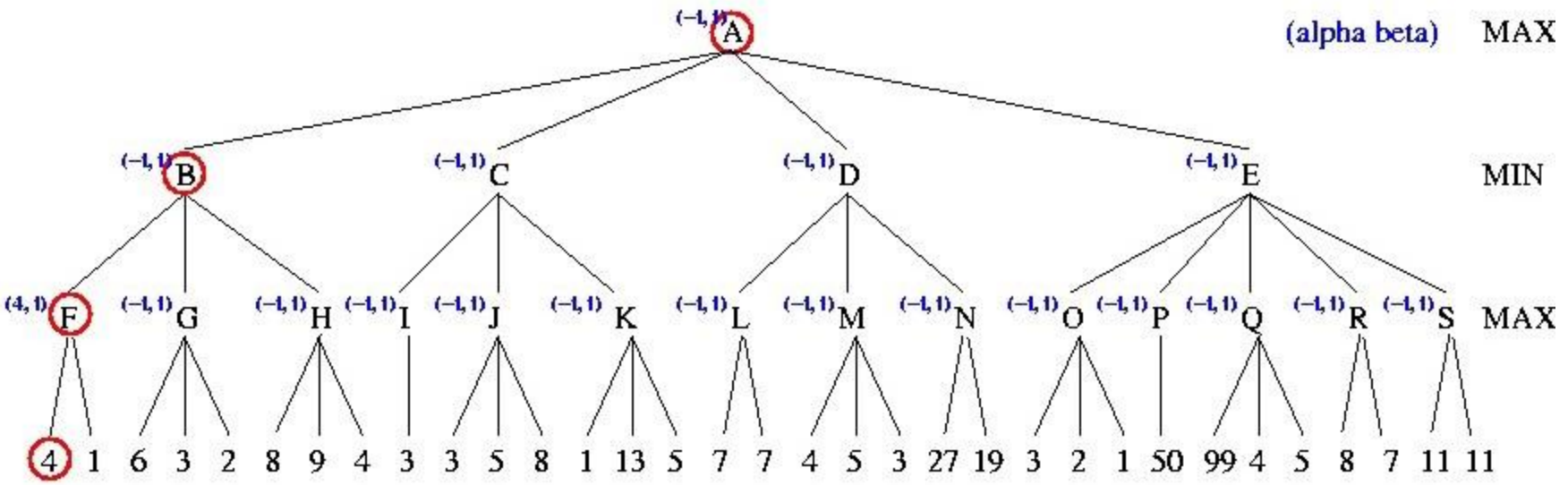
Example


(−1, 1) A
(alpha beta)
MAX
(−1, 1) B
(−1, 1) C
(−1, 1) D
(−1, 1) E
MIN
(−1, 1) F
(−1, 1) G
(−1, 1) H
(−1, 1) I
(−1, 1) J
(−1, 1) K
(−1, 1) L
(−1, 1) M
(−1, 1) N
(−1, 1) O
(−1, 1) P
(−1, 1) Q
(−1, 1) R
(−1, 1) S
MAX
4 1 6 3 2 8 9 4 3 3 5 8 1 13 5 7 7 4 5 3 27 19 3 2 1 50 99 4 5 8 7 11 11

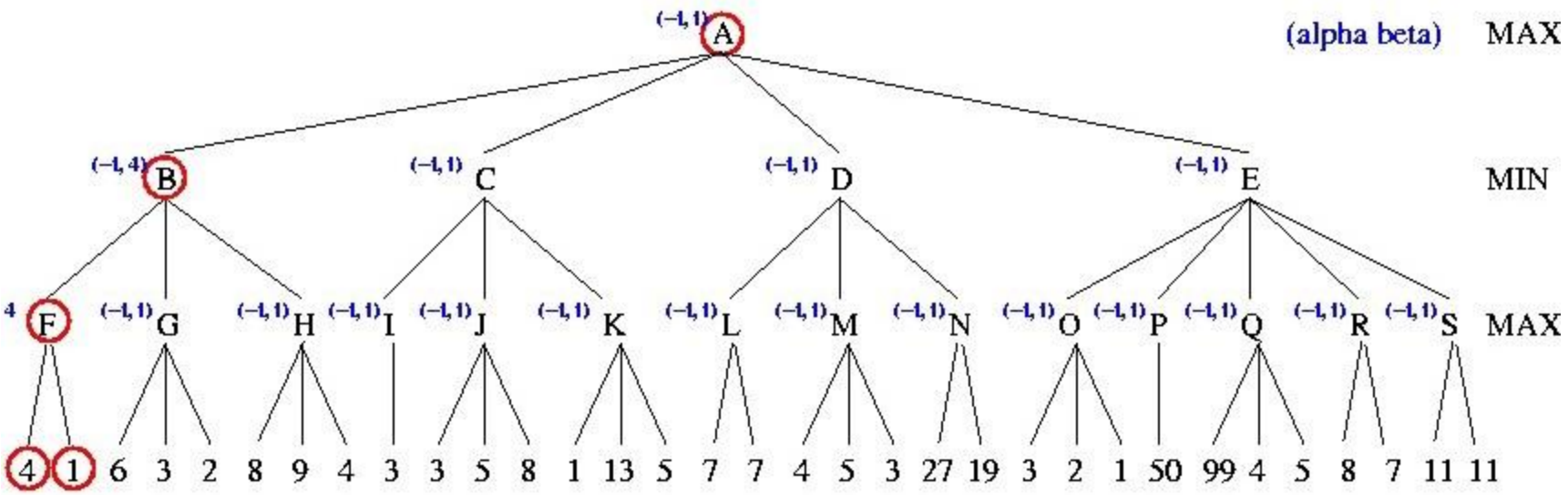
Example


(alpha beta)
MAX
MIN
MAX
A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P
Q
R
S
4 1 6 3 2 8 9 4 3 3 5 8 1 13 5 7 7 4 5 3 27 19 3 2 1 50 99 4 5 8 7 11 11

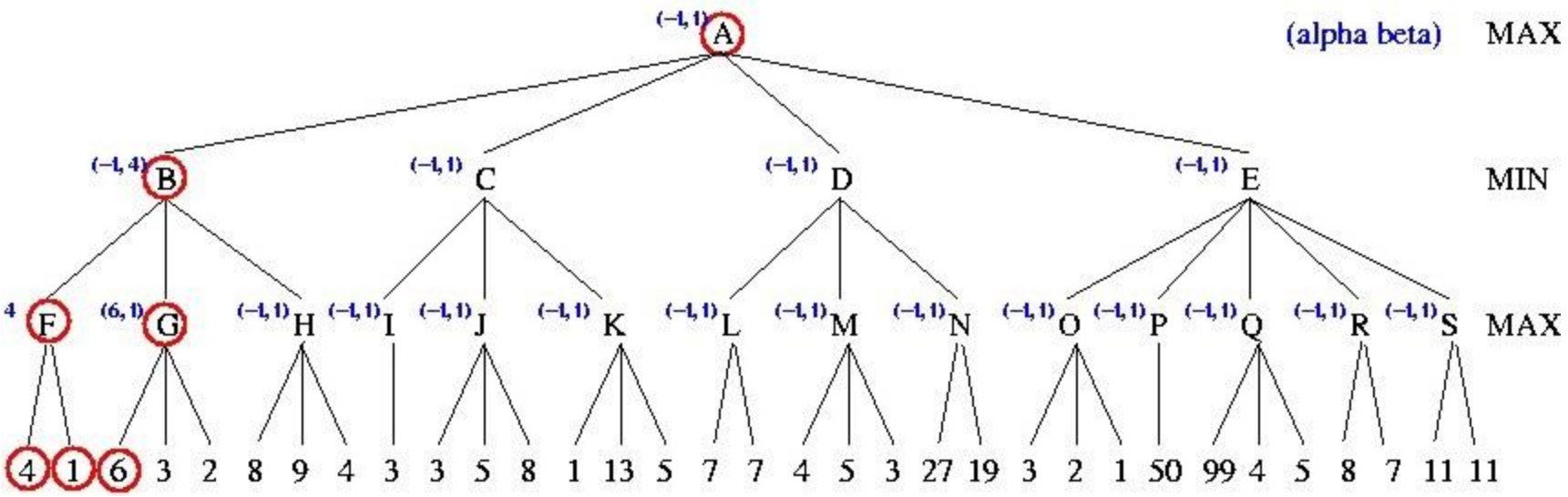
Example


(alpha beta)
MAX
MIN
MAX
A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P
Q
R
S
4 1 6 3 2 8 9 4 3 3 5 8 1 13 5 7 7 4 5 3 27 19 3 2 1 50 99 4 5 8 7 11 11

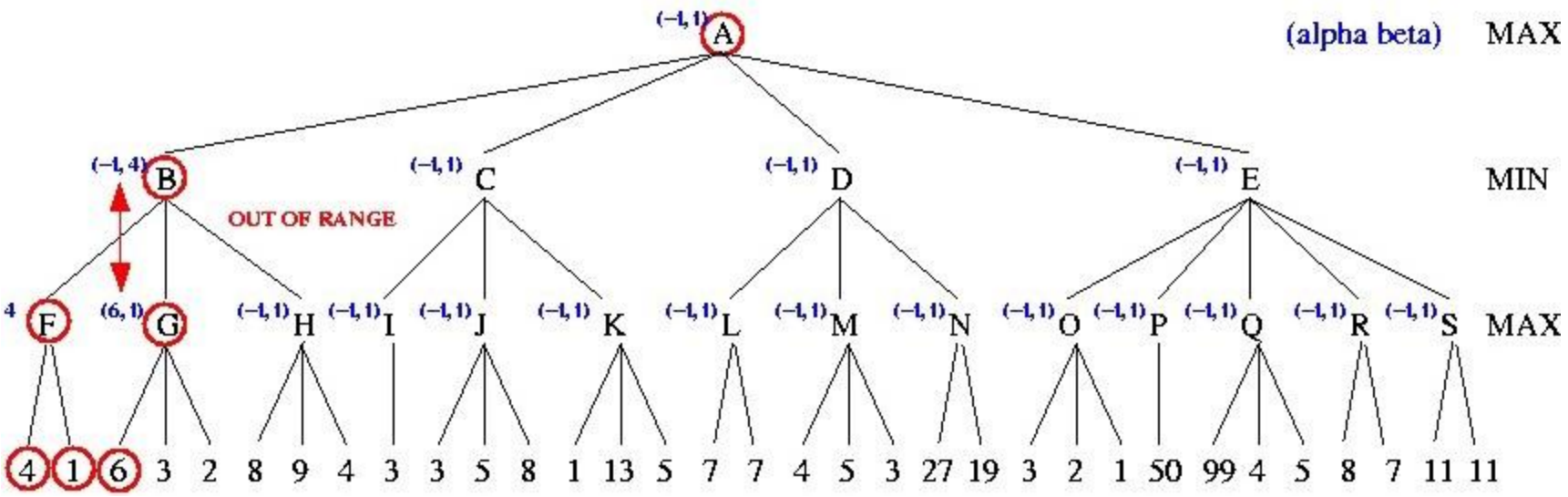
Example


(−1, 1) A
(alpha beta)
MAX
(−1, 4) B
(−1, 1) C
(−1, 1) D
(−1, 1) E
MIN
4 F
(−1, 1) G
(−1, 1) H
(−1, 1) I
(−1, 1) J
(−1, 1) K
(−1, 1) L
(−1, 1) M
(−1, 1) N
(−1, 1) O
(−1, 1) P
(−1, 1) Q
(−1, 1) R
(−1, 1) S
MAX
4 1 6 3 2 8 9 4 3 3 5 8 1 13 5 7 7 4 5 3 27 19 3 2 1 50 99 4 5 8 7 11 11

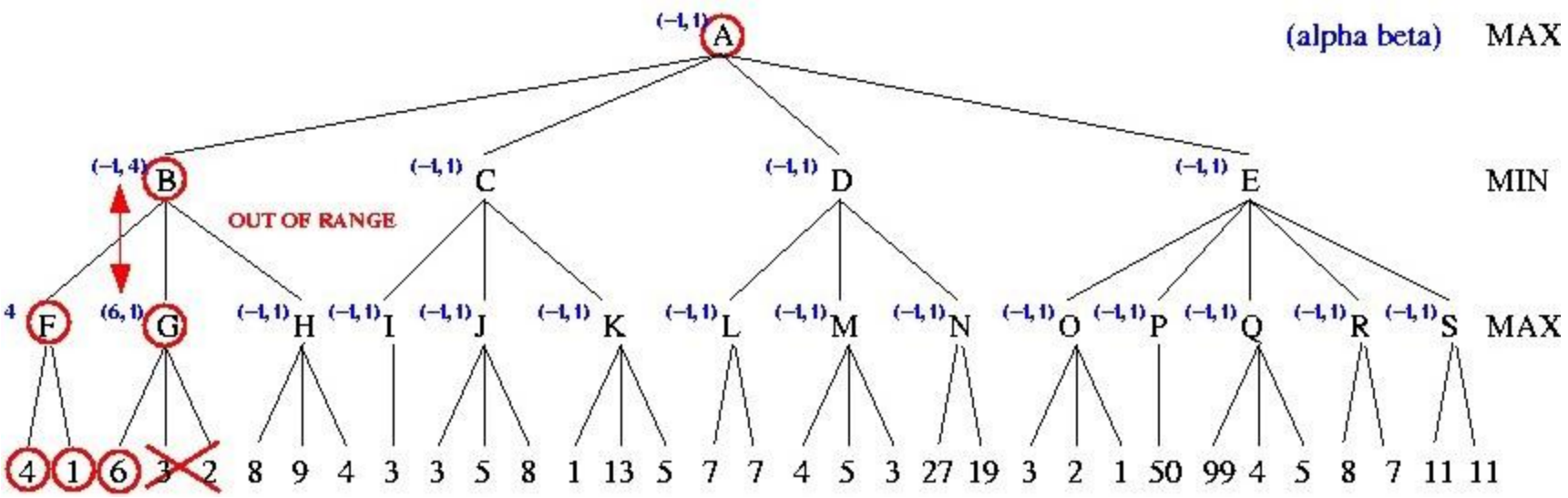
Example


(−1, 1) A
(alpha beta)
MAX
(−1, 4) B
(−1, 1) C
(−1, 1) D
(−1, 1) E
MIN
4 F
(6, 1) G
(−1, 1) H
(−1, 1) I
(−1, 1) J
(−1, 1) K
(−1, 1) L
(−1, 1) M
(−1, 1) N
(−1, 1) O
(−1, 1) P
(−1, 1) Q
(−1, 1) R
(−1, 1) S
MAX
4 1 6 3 2 8 9 4 3 3 5 8 1 13 5 7 7 4 5 3 27 19 3 2 1 50 99 4 5 8 7 11 11

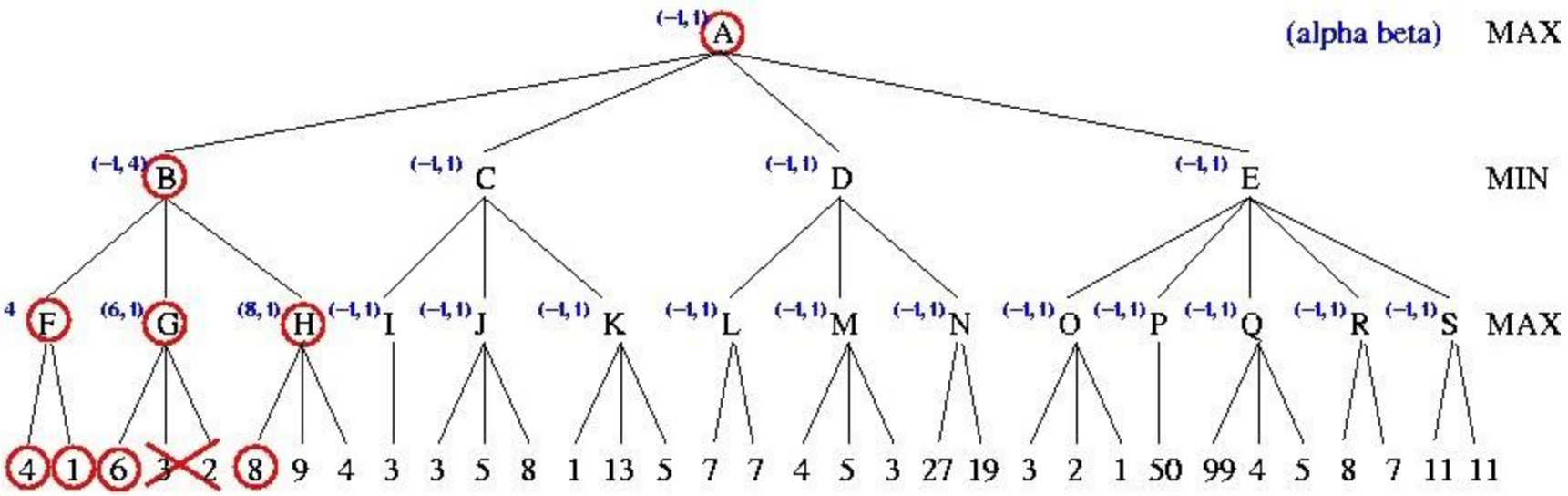
Example


(−1, 1) A
(alpha beta)
MAX
(−1, 4) B
(−1, 1) C
(−1, 1) D
(−1, 1) E
MIN
OUT OF RANGE
4 F
(6, 1) G
(−1, 1) H
(−1, 1) I
(−1, 1) J
(−1, 1) K
(−1, 1) L
(−1, 1) M
(−1, 1) N
(−1, 1) O
(−1, 1) P
(−1, 1) Q
(−1, 1) R
(−1, 1) S
MAX
4 1 6 3 2 8 9 4 3 3 5 8 1 13 5 7 7 4 5 3 27 19 3 2 1 50 99 4 5 8 7 11 11

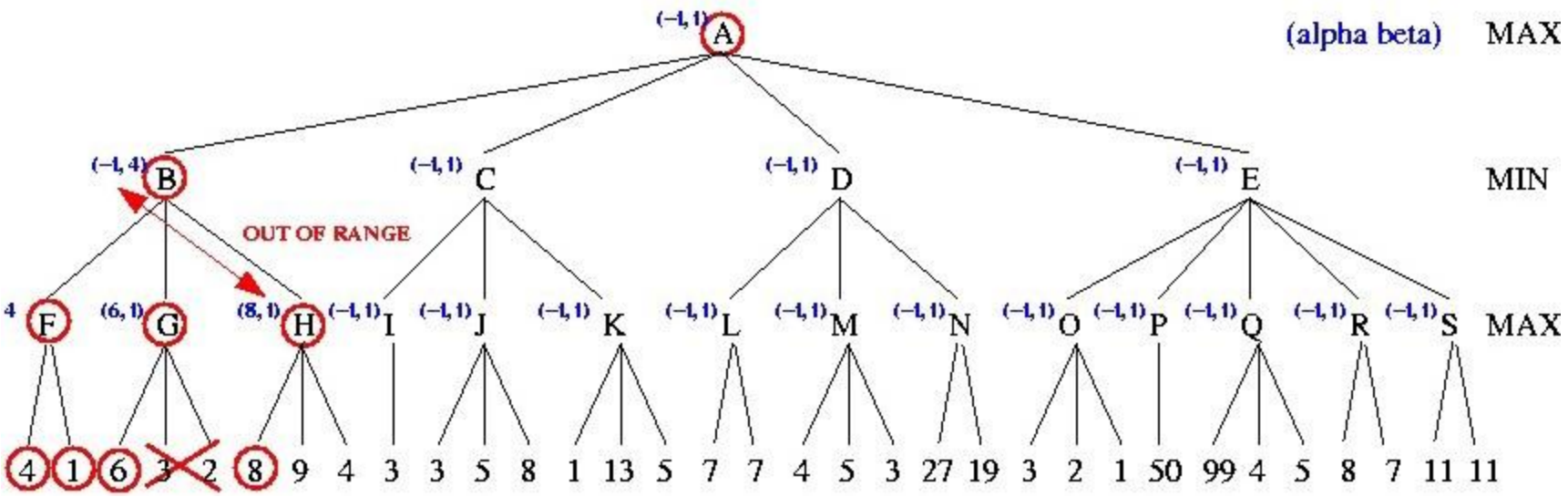
Example


(−1, 1) A
(alpha beta)
MAX
(−1, 4) B
(−1, 1) C
(−1, 1) D
(−1, 1) E
MIN
OUT OF RANGE
4 F
(6, 1) G
(−1, 1) H
(−1, 1) I
(−1, 1) J
(−1, 1) K
(−1, 1) L
(−1, 1) M
(−1, 1) N
(−1, 1) O
(−1, 1) P
(−1, 1) Q
(−1, 1) R
(−1, 1) S
MAX
4 1 6 3 2 8 9 4 3 3 5 8 1 13 5 7 7 4 5 3 27 19 3 2 1 50 99 4 5 8 7 11 11

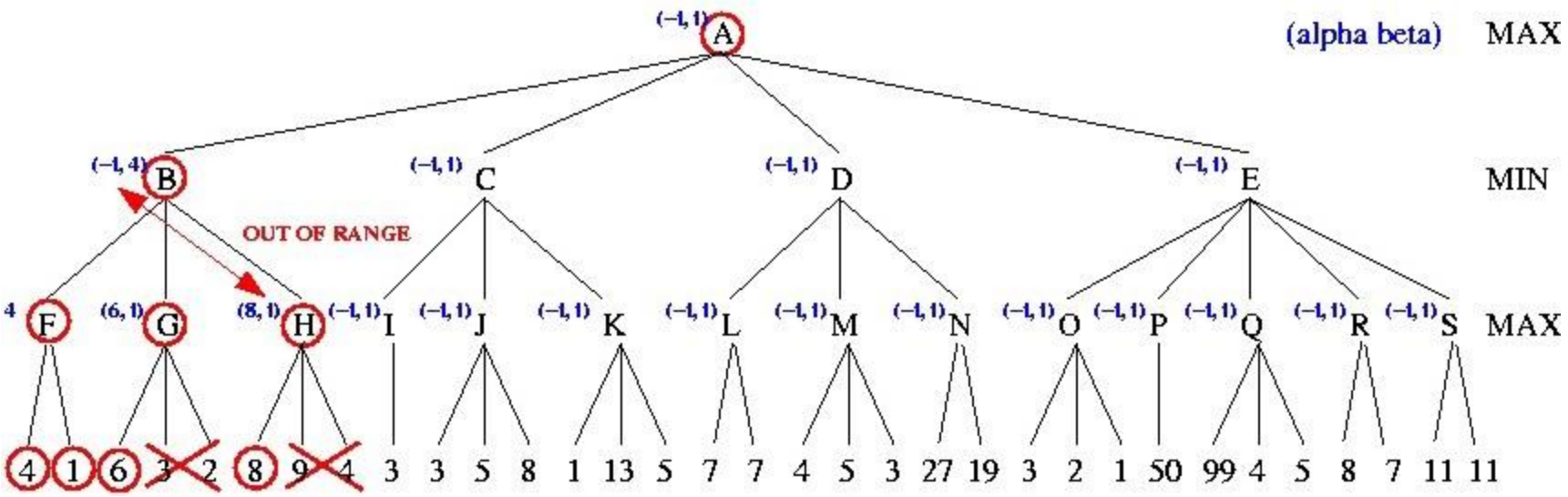
Example


(−1, 1) A
(alpha beta)
MAX
(−1, 4) B
(−1, 1) C
(−1, 1) D
(−1, 1) E
MIN
4 F
(6, 1) G
(8, 1) H
(−1, 1) I
(−1, 1) J
(−1, 1) K
(−1, 1) L
(−1, 1) M
(−1, 1) N
(−1, 1) O
(−1, 1) P
(−1, 1) Q
(−1, 1) R
(−1, 1) S
MAX
4 1 6 3 2 8 9 4 3 3 5 8 1 13 5 7 7 4 5 3 27 19 3 2 1 50 99 4 5 8 7 11 11

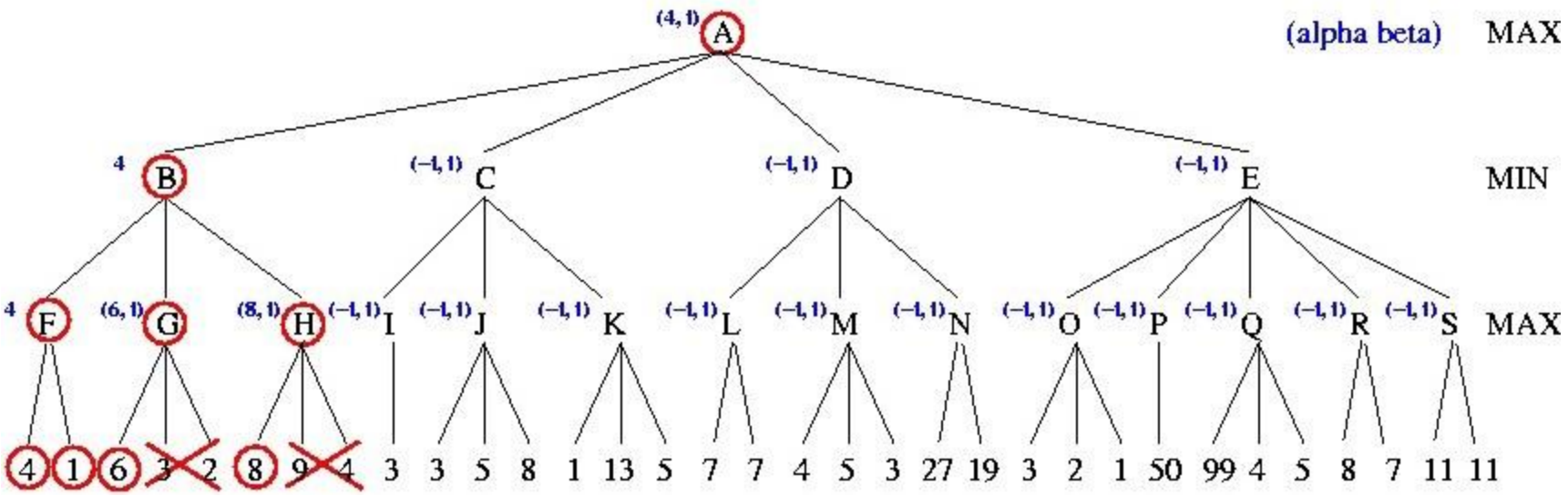
Example


(alpha beta)
MAX
MIN
MAX
OUT OF RANGE
A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P
Q
R
S
4 1 6 3 2 8 9 4 3 3 5 8 1 13 5 7 7 4 5 3 27 19 3 2 1 50 99 4 5 8 7 11 11

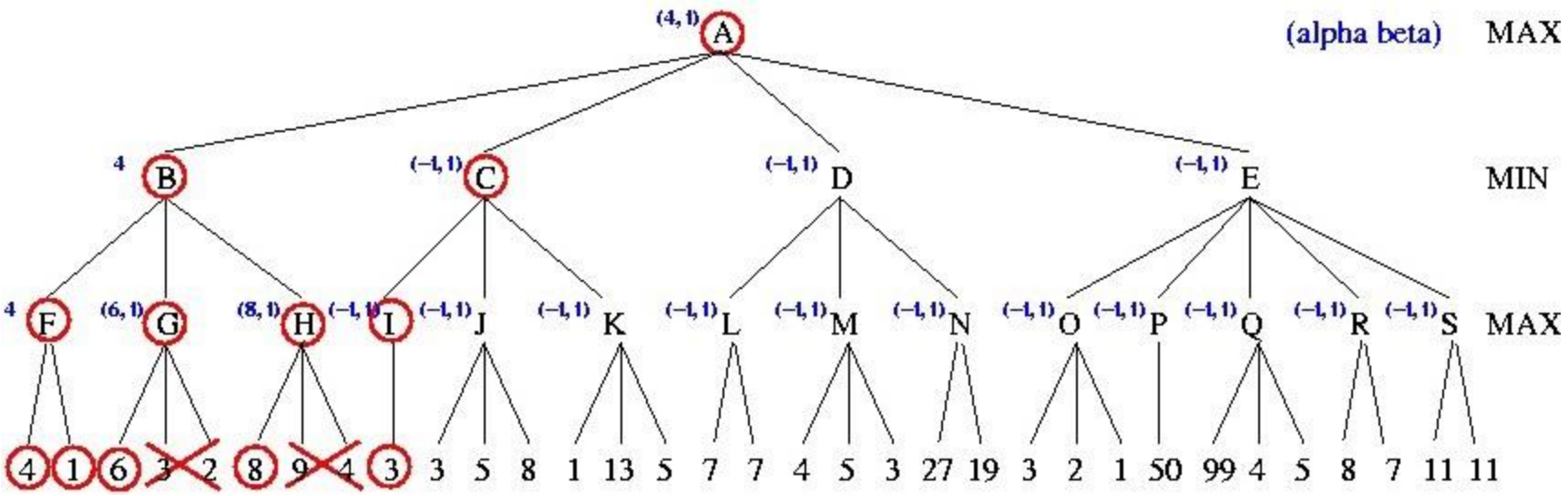
Example


(alpha beta)
MAX
MIN
MAX
OUT OF RANGE
A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P
Q
R
S
4 1 6 3 2 8 9 4 3 3 5 8 1 13 5 7 7 4 5 3 27 19 3 2 1 50 99 4 5 8 7 11 11

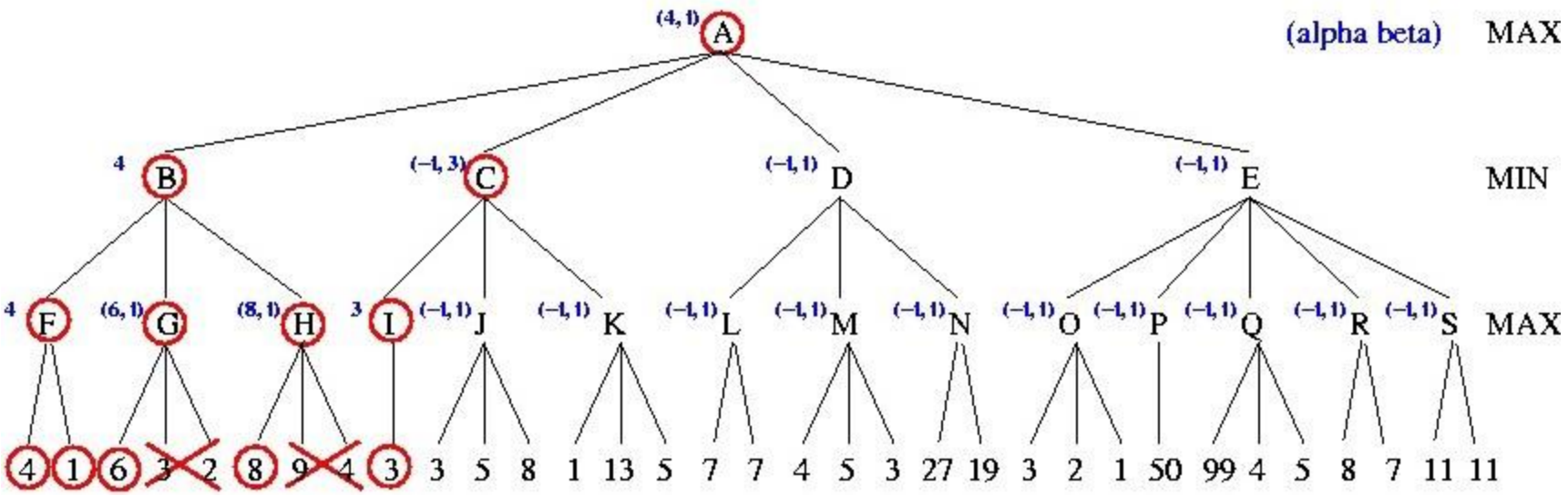
Example


(4, 1) A
(alpha beta)
MAX
4 B
(−1, 1) C
(−1, 1) D
(−1, 1) E
MIN
4 F
(6, 1) G
(8, 1) H
(−1, 1) I
(−1, 1) J
(−1, 1) K
(−1, 1) L
(−1, 1) M
(−1, 1) N
(−1, 1) O
(−1, 1) P
(−1, 1) Q
(−1, 1) R
(−1, 1) S
MAX
4 1 6 3 2 8 9 4 3 3 5 8 1 13 5 7 7 4 5 3 27 19 3 2 1 50 99 4 5 8 7 11 11

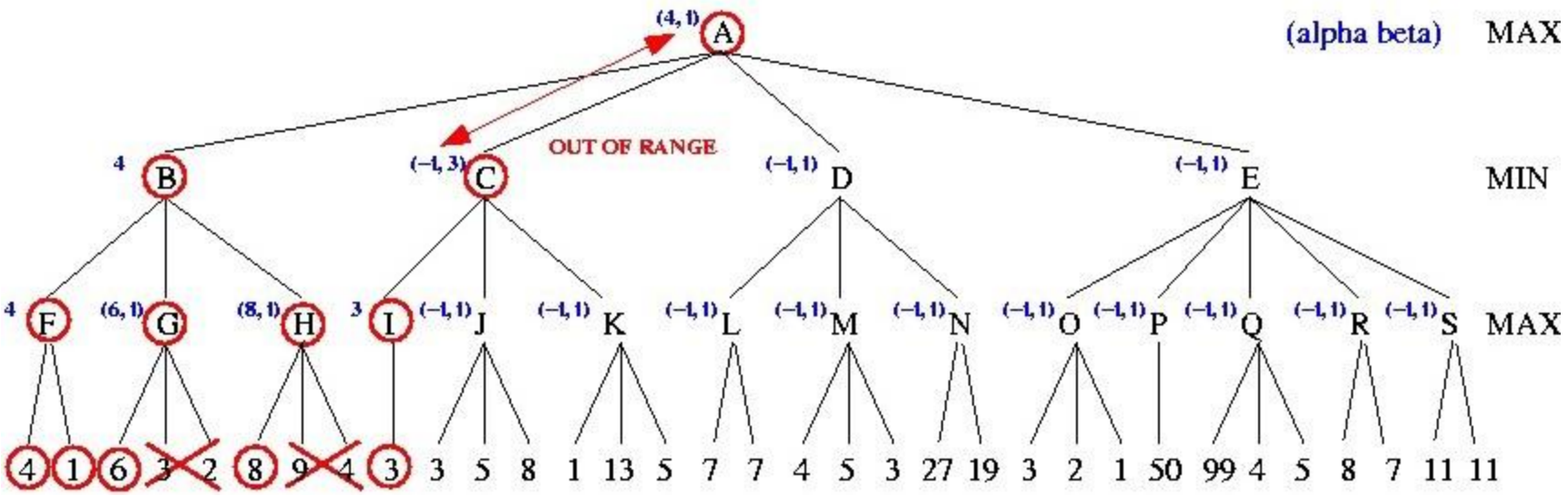
Example


(4, 1) A
(alpha beta)
MAX
4 B
(−1, 1) C
(−1, 1) D
(−1, 1) E
MIN
4 F
(6, 1) G
(8, 1) H
(−1, 1) I
(−1, 1) J
(−1, 1) K
(−1, 1) L
(−1, 1) M
(−1, 1) N
(−1, 1) O
(−1, 1) P
(−1, 1) Q
(−1, 1) R
(−1, 1) S
MAX
4 1 6 3 2 8 9 4 3 3 5 8 1 13 5 7 7 4 5 3 27 19 3 2 1 50 99 4 5 8 7 11 11

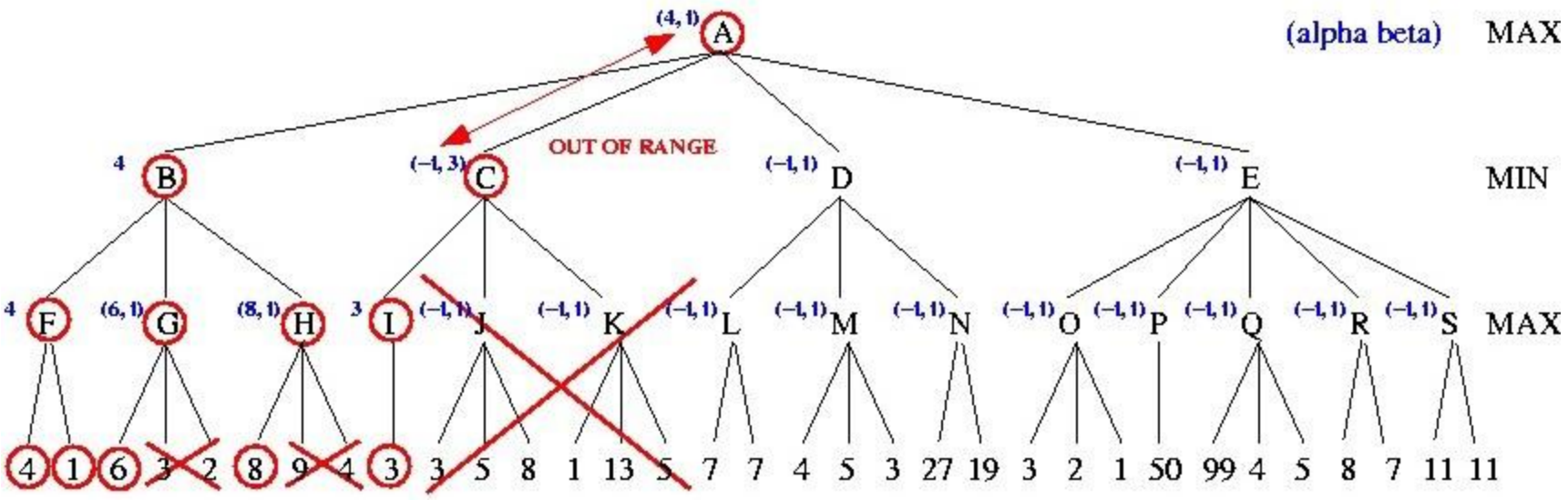
Example


(4, 1) A
(alpha beta)
MAX
4 B
(−1, 3) C
(−1, 1) D
(−1, 1) E
MIN
4 F
(6, 1) G
(8, 1) H
3 I
(−1, 1) J
(−1, 1) K
(−1, 1) L
(−1, 1) M
(−1, 1) N
(−1, 1) O
(−1, 1) P
(−1, 1) Q
(−1, 1) R
(−1, 1) S
MAX
4 1 6 3 2 8 9 4 3 3 5 8 1 13 5 7 7 4 5 3 27 19 3 2 1 50 99 4 5 8 7 11 11

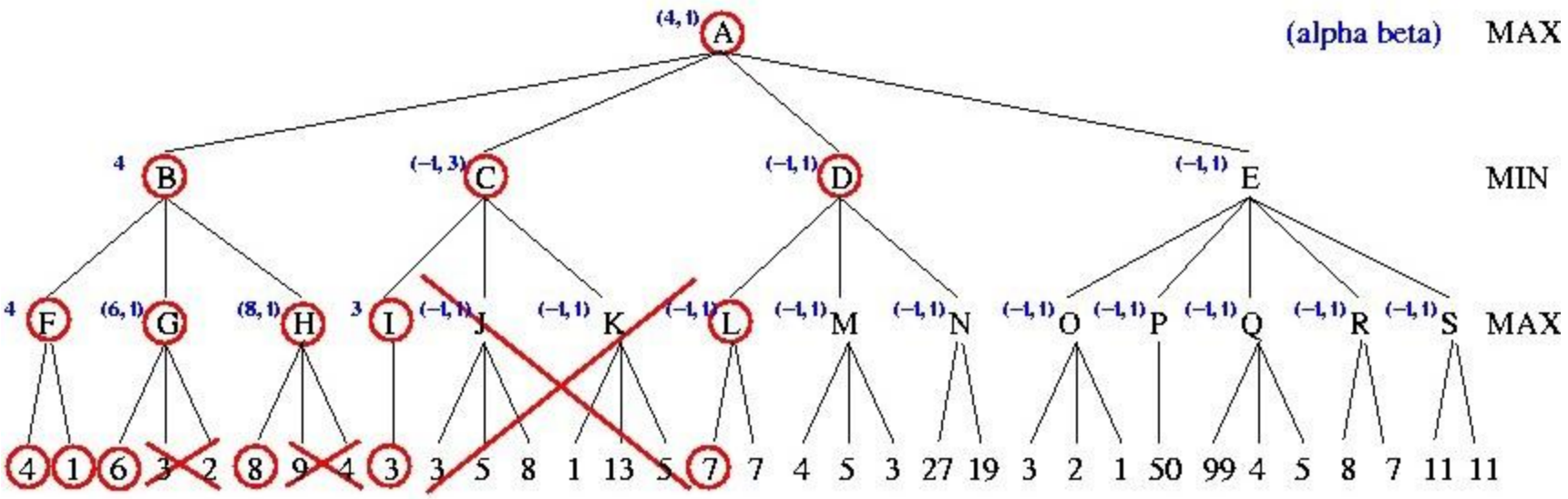
Example


(4, 1) A
(alpha beta)
MAX
4 B
(−1, 3) C
OUT OF RANGE
(−1, 1) D
(−1, 1) E
MIN
4 F
(6, 1) G
(8, 1) H
3 I
(−1, 1) J
(−1, 1) K
(−1, 1) L
(−1, 1) M
(−1, 1) N
(−1, 1) O
(−1, 1) P
(−1, 1) Q
(−1, 1) R
(−1, 1) S
MAX
4 1 6 3 2 8 9 4 3 3 5 8 1 13 5 7 7 4 5 3 27 19 3 2 1 50 99 4 5 8 7 11 11

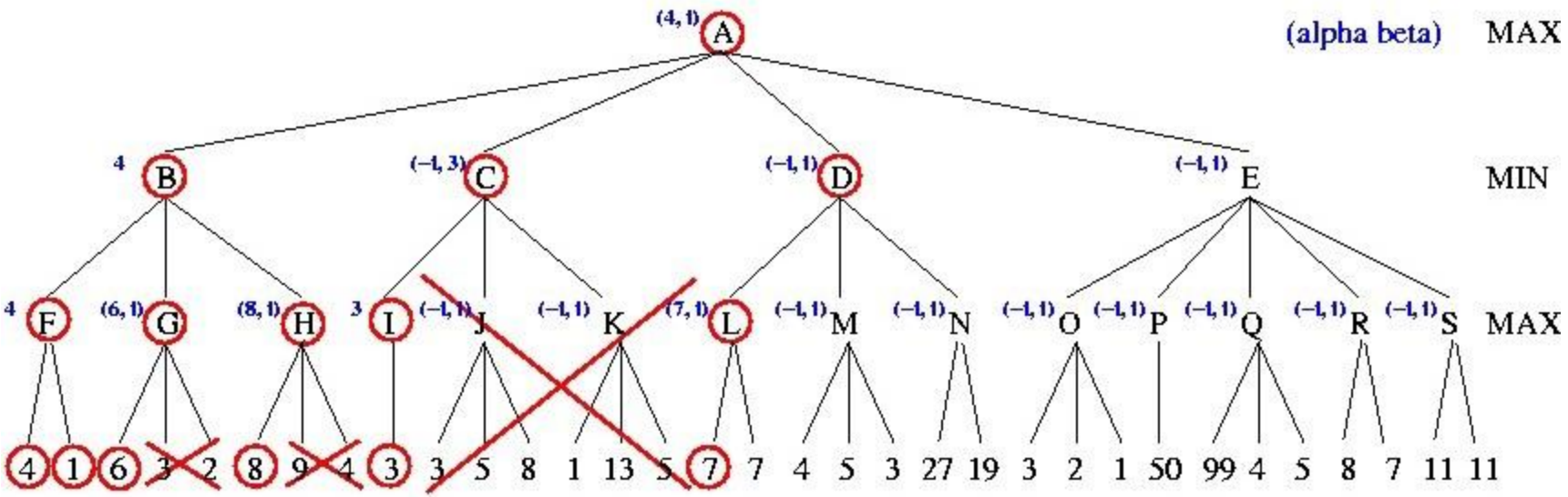
Example


(alpha beta)
MAX
MIN
MAX
OUT OF RANGE
(4, 1) A
4 B
(−1, 3) C
(−1, 1) D
(−1, 1) E
4 F
(6, 1) G
(8, 1) H
3 I
(−1, 1) J
(−1, 1) K
(−1, 1) L
(−1, 1) M
(−1, 1) N
(−1, 1) O
(−1, 1) P
(−1, 1) Q
(−1, 1) R
(−1, 1) S
4 1 6 3 2 8 9 4 3 3 5 8 1 13 5 7 7 4 5 3 27 19 3 2 1 50 99 4 5 8 7 11 11

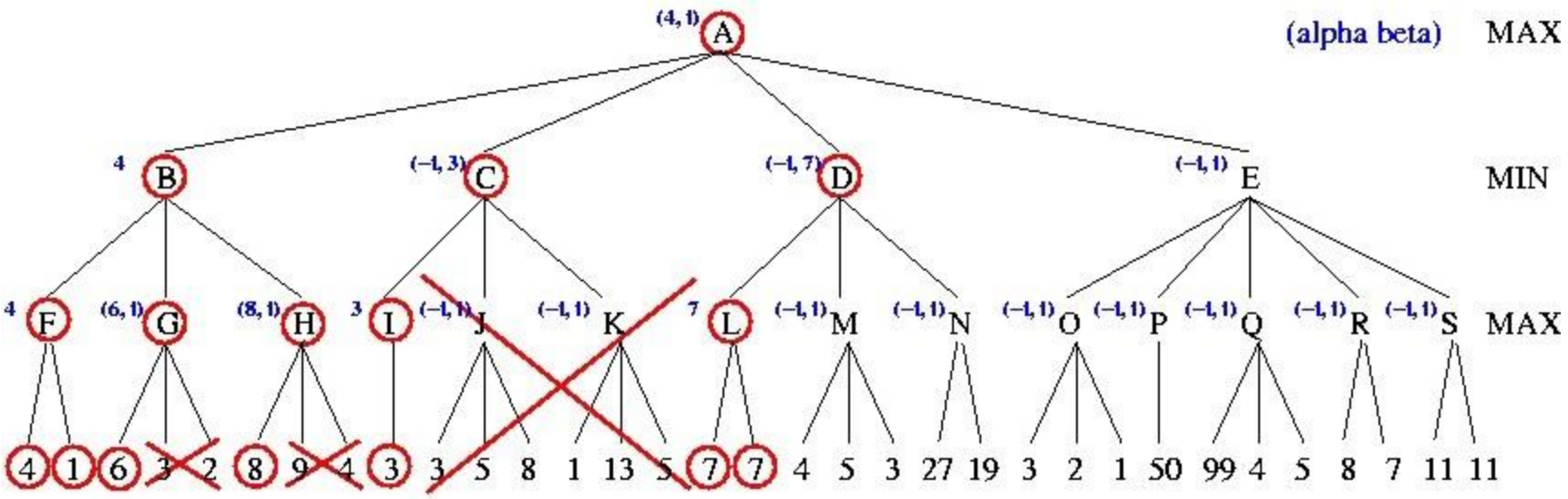
Example


(alpha beta)
MAX
MIN
MAX
A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P
Q
R
S
4 1 6 3 2 8 9 4 3 3 5 8 1 13 5 7 7 4 5 3 27 19 3 2 1 50 99 4 5 8 7 11 11

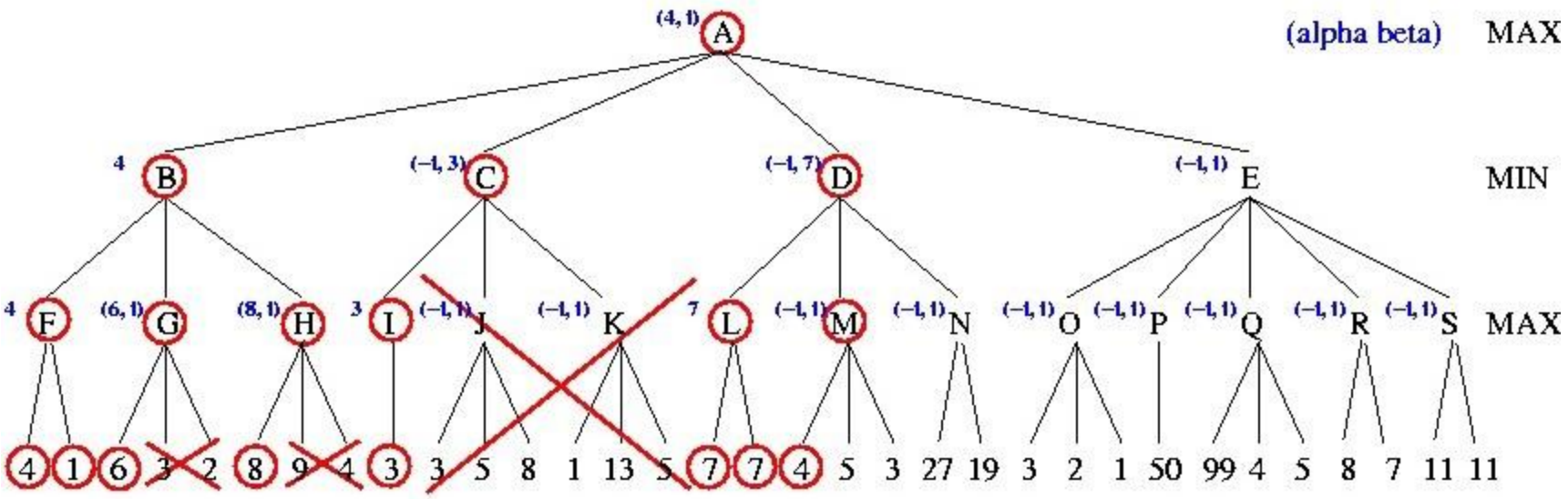
Example


(4, 1) A
(alpha beta)
MAX
4 B
(−1, 3) C
(−1, 1) D
(−1, 1) E
MIN
4 F
(6, 1) G
(8, 1) H
3 I
(−1, 1) J
(−1, 1) K
(7, 1) L
(−1, 1) M
(−1, 1) N
(−1, 1) O
(−1, 1) P
(−1, 1) Q
(−1, 1) R
(−1, 1) S
MAX
4 1 6 3 2 8 9 4 3 3 5 8 1 13 5 7 7 4 5 3 27 19 3 2 1 50 99 4 5 8 7 11 11

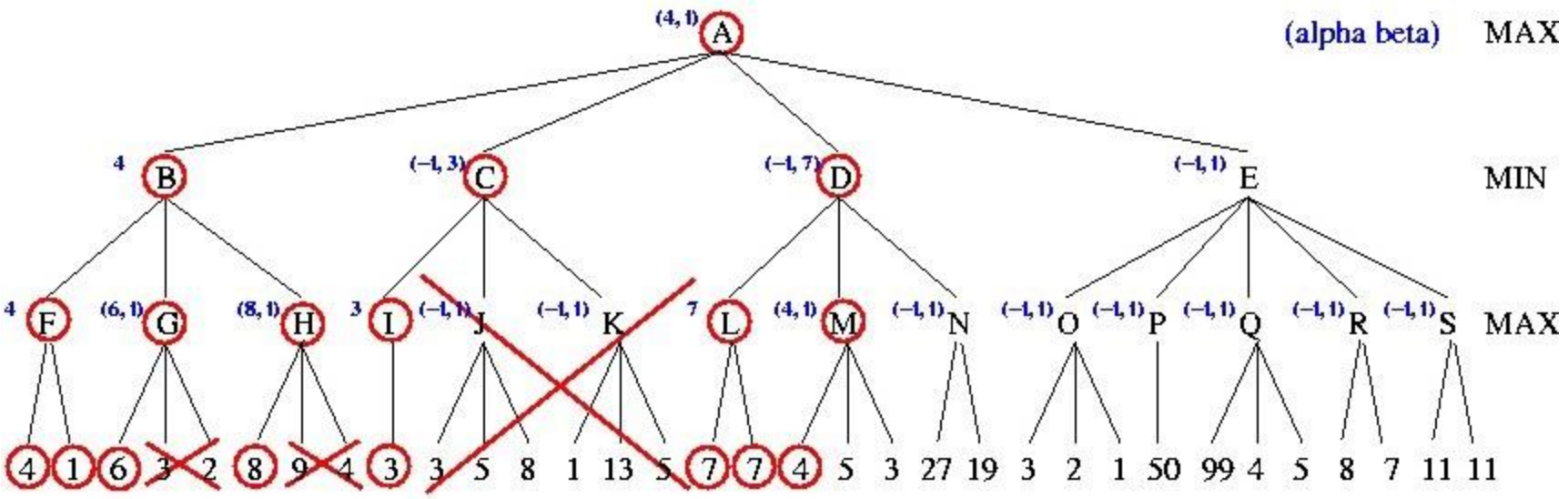
Example


(4, 1) A
(alpha beta)
MAX
4 B
(−1, 3) C
(−1, 7) D
(−1, 1) E
MIN
4 F
(6, 1) G
(8, 1) H
3 I
(−1, 1) J
(−1, 1) K
7 L
(−1, 1) M
(−1, 1) N
(−1, 1) O
(−1, 1) P
(−1, 1) Q
(−1, 1) R
(−1, 1) S
MAX
4 1 6 3 2 8 9 4 3 3 5 8 1 13 5 7 7 4 5 3 27 19 3 2 1 50 99 4 5 8 7 11 11

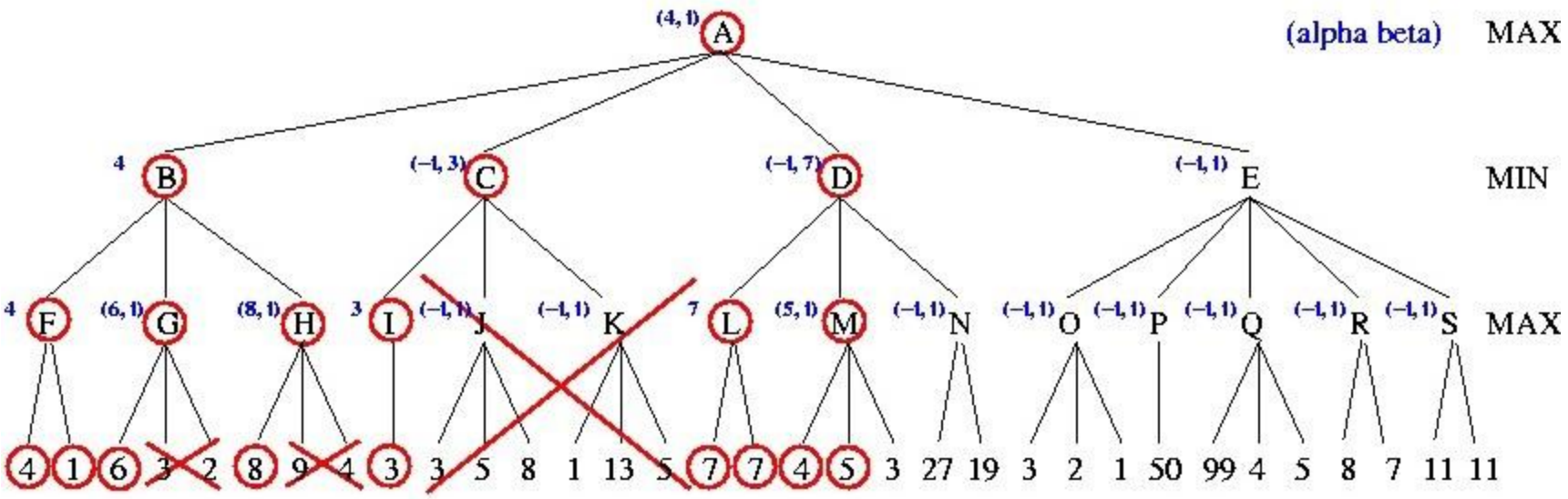
Example


(4, 1) A
(alpha beta)
MAX
4 B
(−1, 3) C
(−1, 7) D
(−1, 1) E
MIN
4 F
(6, 1) G
(8, 1) H
3 I
(−1, 1) J
(−1, 1) K
7 L
(−1, 1) M
(−1, 1) N
(−1, 1) O
(−1, 1) P
(−1, 1) Q
(−1, 1) R
(−1, 1) S
MAX
4 1 6 3 2 8 9 4 3 3 5 8 1 13 5 7 7 4 5 3 27 19 3 2 1 50 99 4 5 8 7 11 11

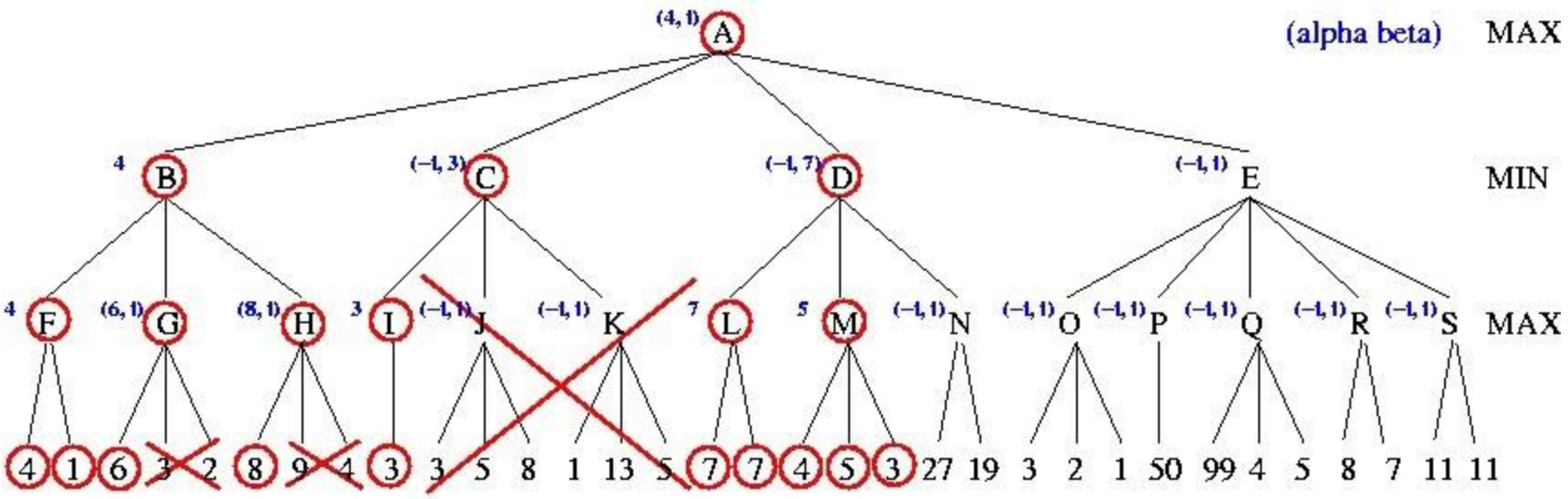
Example


(alpha beta)
MAX
MIN
MAX
A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P
Q
R
S
4
(4, 1)
(−1, 3)
(−1, 7)
(−1, 1)
(6, 1)
(8, 1)
3
7
4 1 6 3 2 8 9 4 3 3 5 8 1 13 5 7 7 4 5 3 27 19 3 2 1 50 99 4 5 8 7 11 11

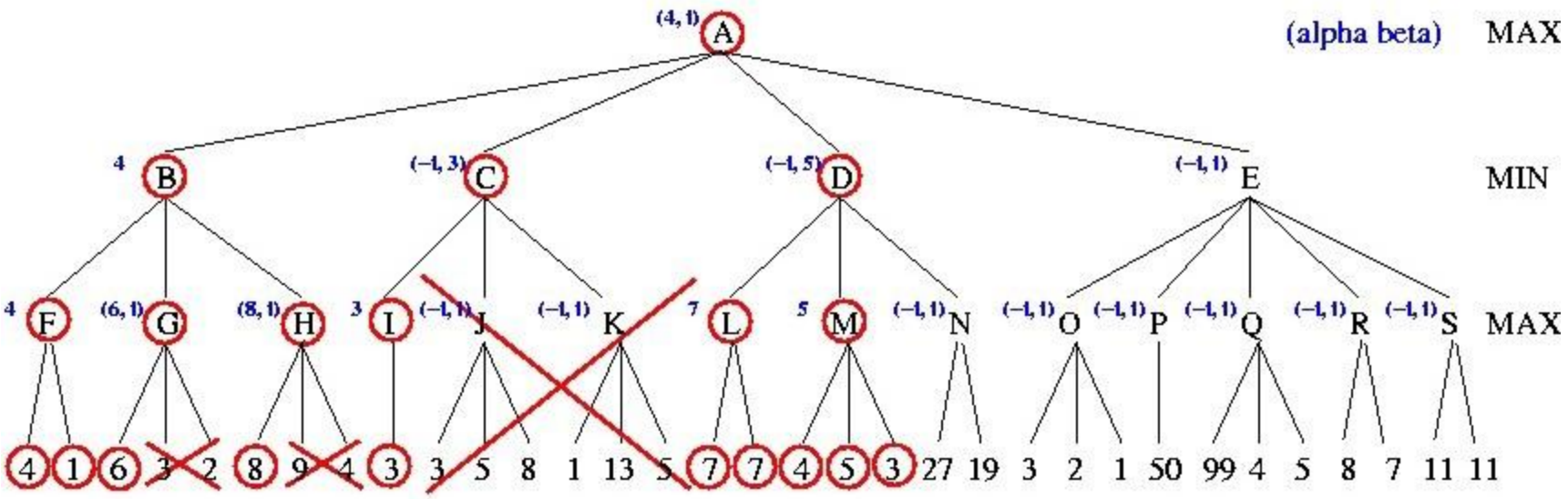
Example


(4, 1) A
(alpha beta)
MAX
4 B
(−1, 3) C
(−1, 7) D
(−1, 1) E
MIN
4 F
(6, 1) G
(8, 1) H
3 I
(−1, 1) J
(−1, 1) K
7 L
(5, 1) M
(−1, 1) N
(−1, 1) O
(−1, 1) P
(−1, 1) Q
(−1, 1) R
(−1, 1) S
MAX
4 1 6 3 2 8 9 4 3 3 5 8 1 13 5 7 7 4 5 3 27 19 3 2 1 50 99 4 5 8 7 11 11

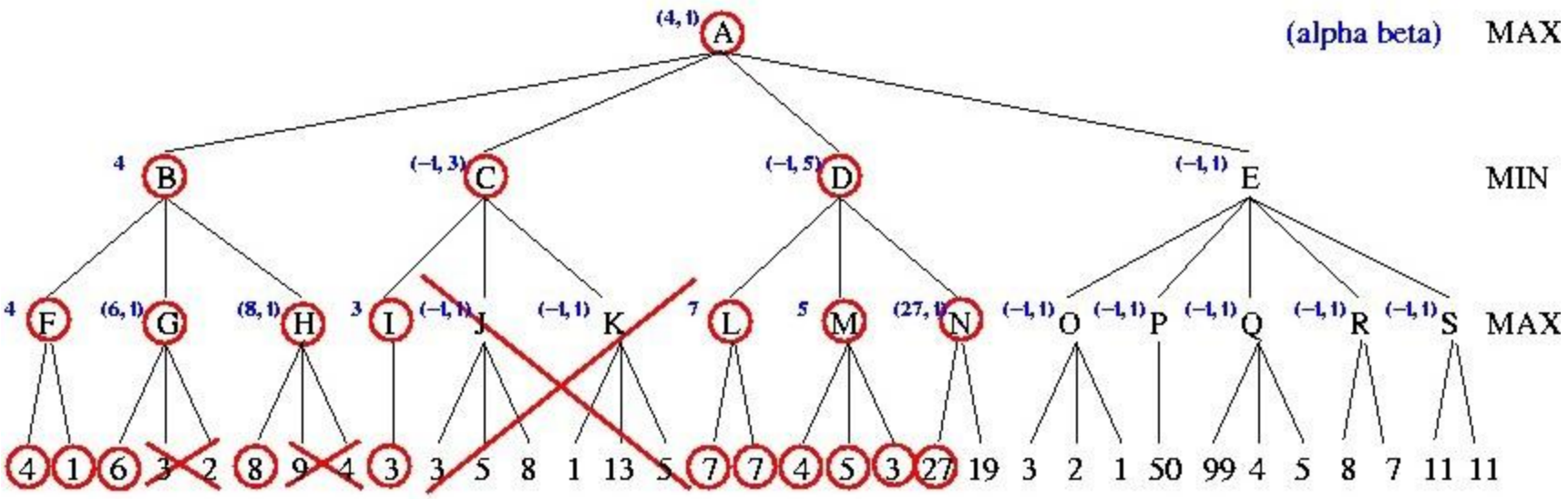
Example


(alpha beta)
MAX
MIN
MAX
A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P
Q
R
S
(4, 1)
4
(−1, 3)
(−1, 7)
(−1, 1)
(6, 1)
(8, 1)
3
5
7
4 1 6 3 2 8 9 4 3 3 5 8 1 13 5 7 7 4 5 3 27 19 3 2 1 50 99 4 5 8 7 11 11

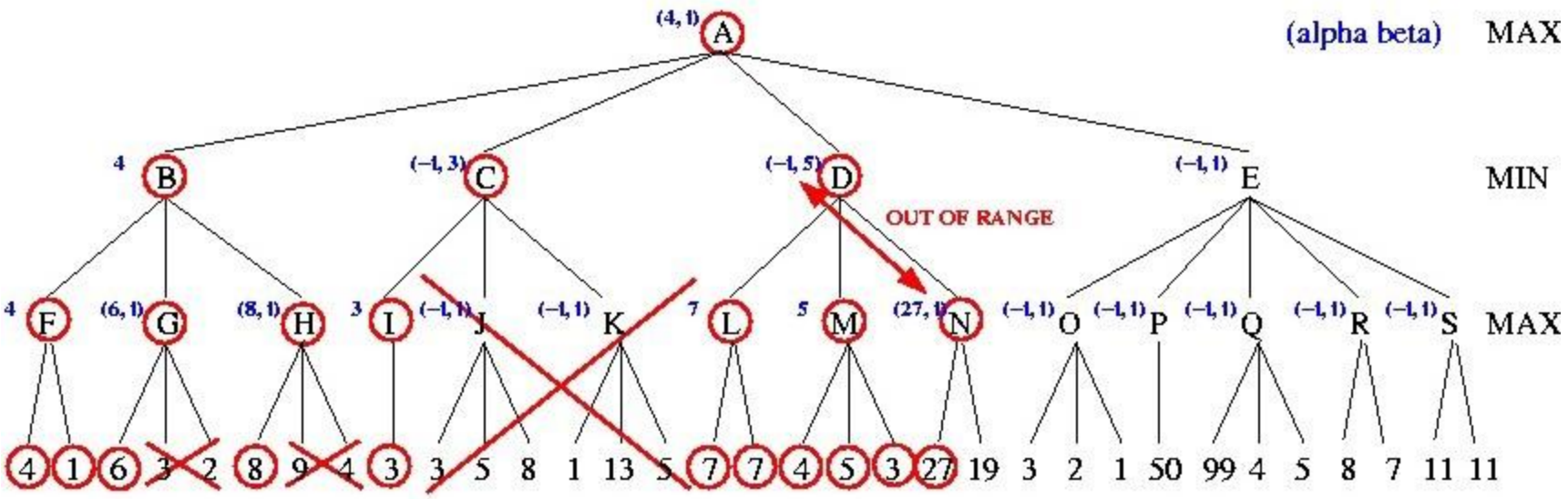
Example


(alpha beta)
MAX
MIN
MAX
A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P
Q
R
S
(4, 1)
4
(−1, 3)
(−1, 5)
(−1, 1)
(6, 1)
(8, 1)
3
7
5
4 1 6 3 2 8 9 4 3 3 5 8 1 13 5 7 7 4 5 3 27 19 3 2 1 50 99 4 5 8 7 11 11

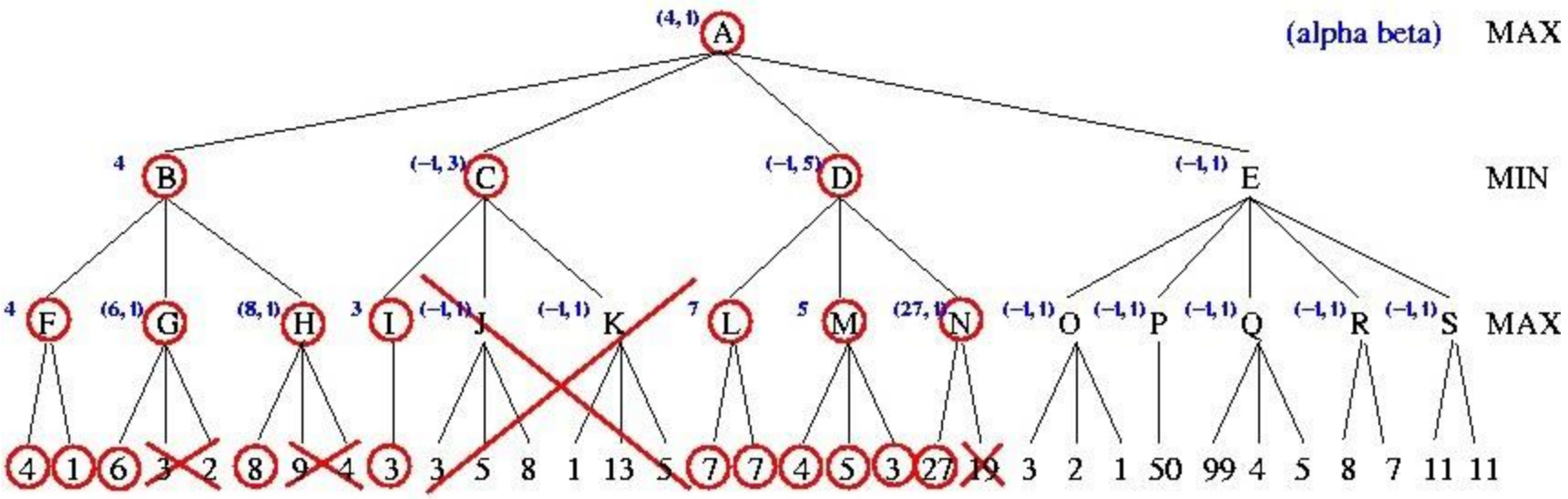
Example


(alpha beta)
MAX
MIN
MAX
A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P
Q
R
S
4
(4, 1)
(−1, 3)
(−1, 5)
(−1, 1)
(6, 1)
(8, 1)
3
(−1, 1)
7
5
(27, 1)
4 1 6 3 2 8 9 4 3 3 5 8 1 13 5 7 7 4 5 3 27 19 3 2 1 50 99 4 5 8 7 11 11

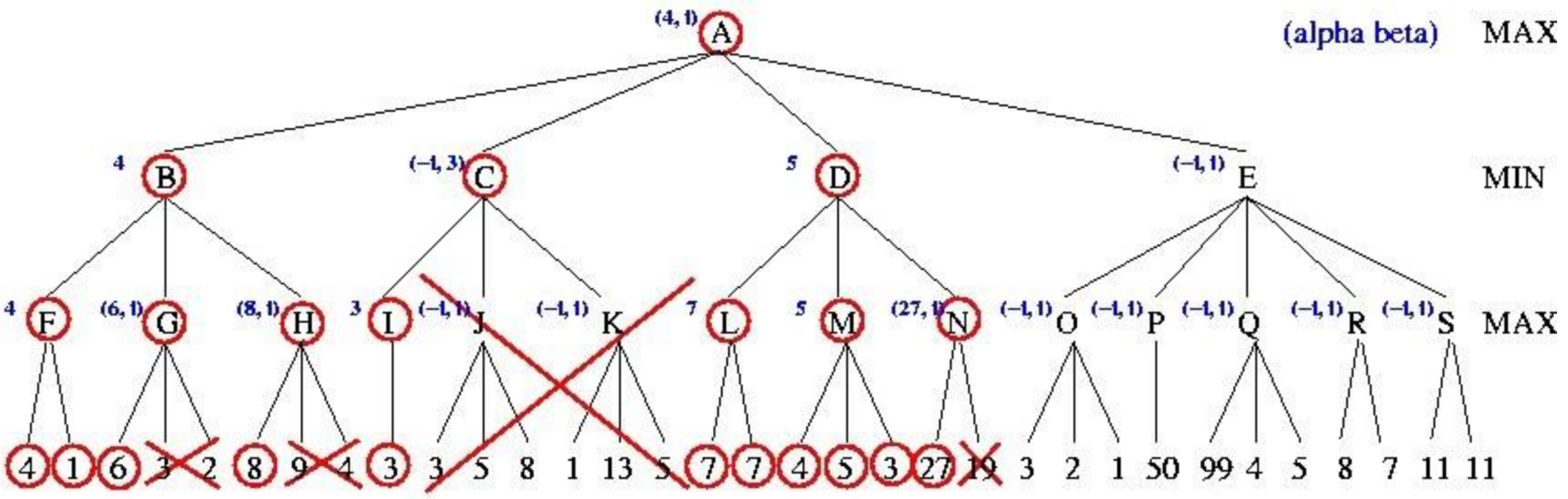
Example


(4, 1) A
(alpha beta)
MAX
4 B
(−1, 3) C
(−1, 5) D
(−1, 1) E
MIN
OUT OF RANGE
4 F
(6, 1) G
(8, 1) H
3 I
(−1, 1) J
(−1, 1) K
7 L
5 M
(27, 1) N
(−1, 1) O
(−1, 1) P
(−1, 1) Q
(−1, 1) R
(−1, 1) S
MAX
4 1 6 3 2 8 9 4 3 3 5 8 1 13 5 7 7 4 5 3 27 19 3 2 1 50 99 4 5 8 7 11 11

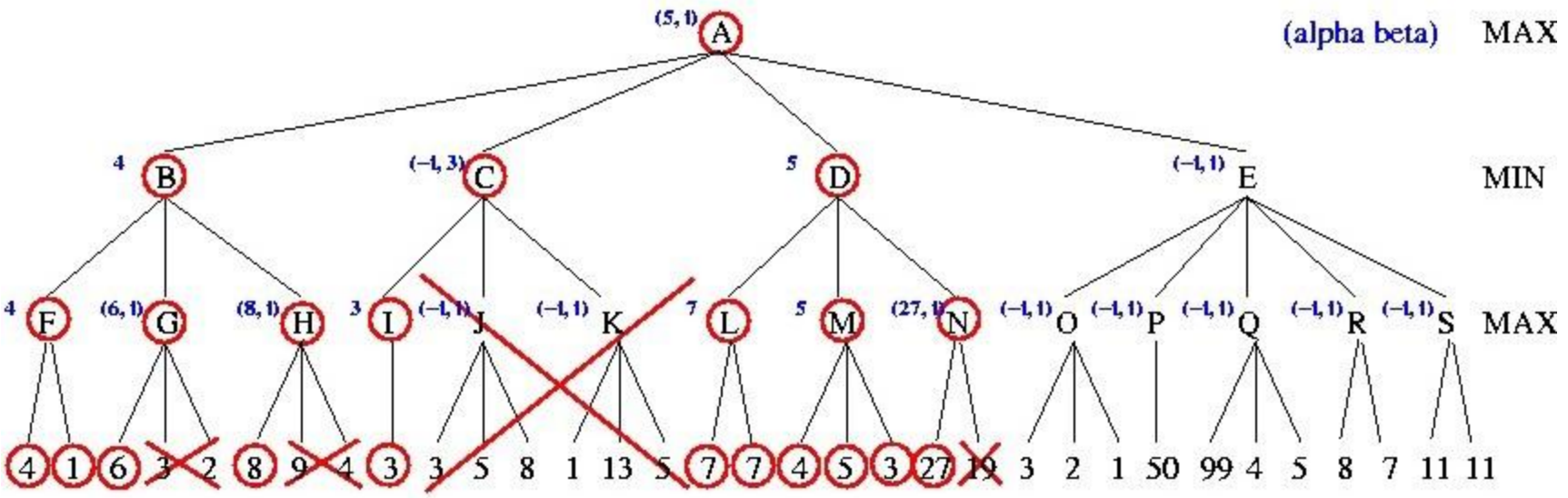
Example


(4, 1) A
(alpha beta)
MAX
4 B
(−1, 3) C
(−1, 5) D
(−1, 1) E
MIN
4 F
(6, 1) G
(8, 1) H
3 I
(−1, 1) J
(−1, 1) K
7 L
5 M
(27, 1) N
(−1, 1) O
(−1, 1) P
(−1, 1) Q
(−1, 1) R
(−1, 1) S
MAX
4 1 6 3 2 8 9 4 3 3 5 8 1 13 5 7 7 4 5 3 27 19 3 2 1 50 99 4 5 8 7 11 11

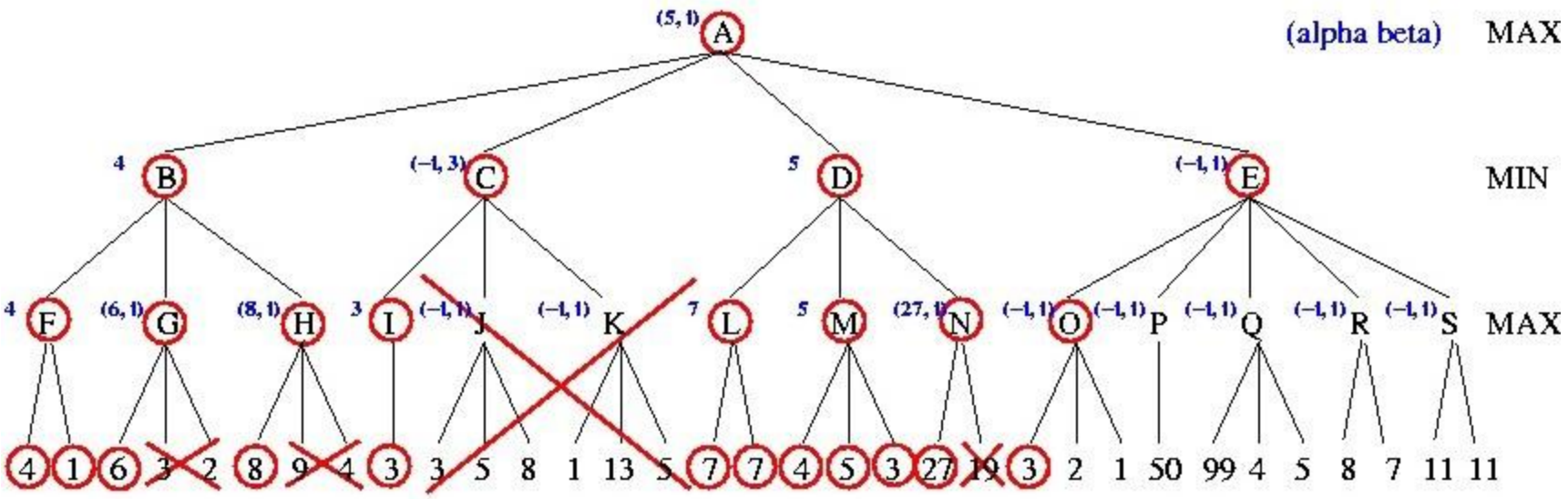
Example


(4, 1) A
(alpha beta)
MAX
4 B
(−1, 3) C
5 D
(−1, 1) E
MIN
4 F
(6, 1) G
(8, 1) H
3 I
(−1, 1) J
(−1, 1) K
7 L
5 M
(27, 1) N
(−1, 1) O
(−1, 1) P
(−1, 1) Q
(−1, 1) R
(−1, 1) S
MAX
4 1 6 3 2 8 9 4 3 3 5 8 1 13 5 7 7 4 5 3 27 19 3 2 1 50 99 4 5 8 7 11 11

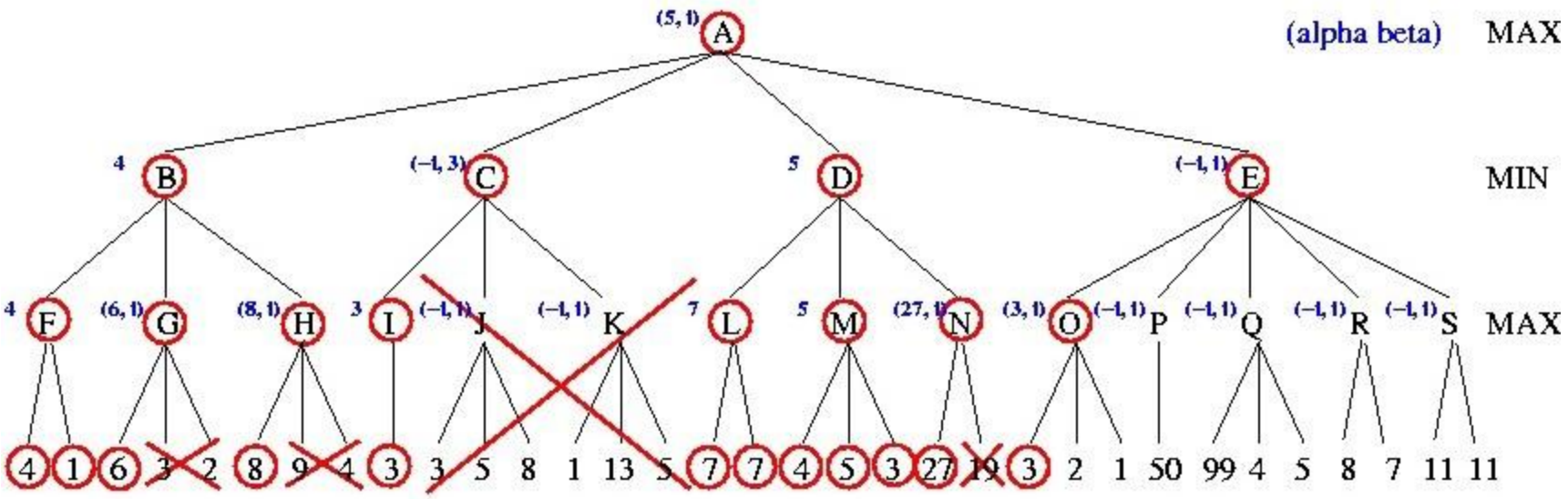
Example


(alpha beta)
MAX
MIN
MAX
(5, 1) A
4 B
(−1, 3) C
5 D
(−1, 1) E
4 F
(6, 1) G
(8, 1) H
3 I
(−1, 1) J
(−1, 1) K
7 L
5 M
(27, 1) N
(−1, 1) O
(−1, 1) P
(−1, 1) Q
(−1, 1) R
(−1, 1) S
4 1 6 3 2 8 9 4 3 3 5 8 1 13 5 7 7 4 5 3 27 19 3 2 1 50 99 4 5 8 7 11 11

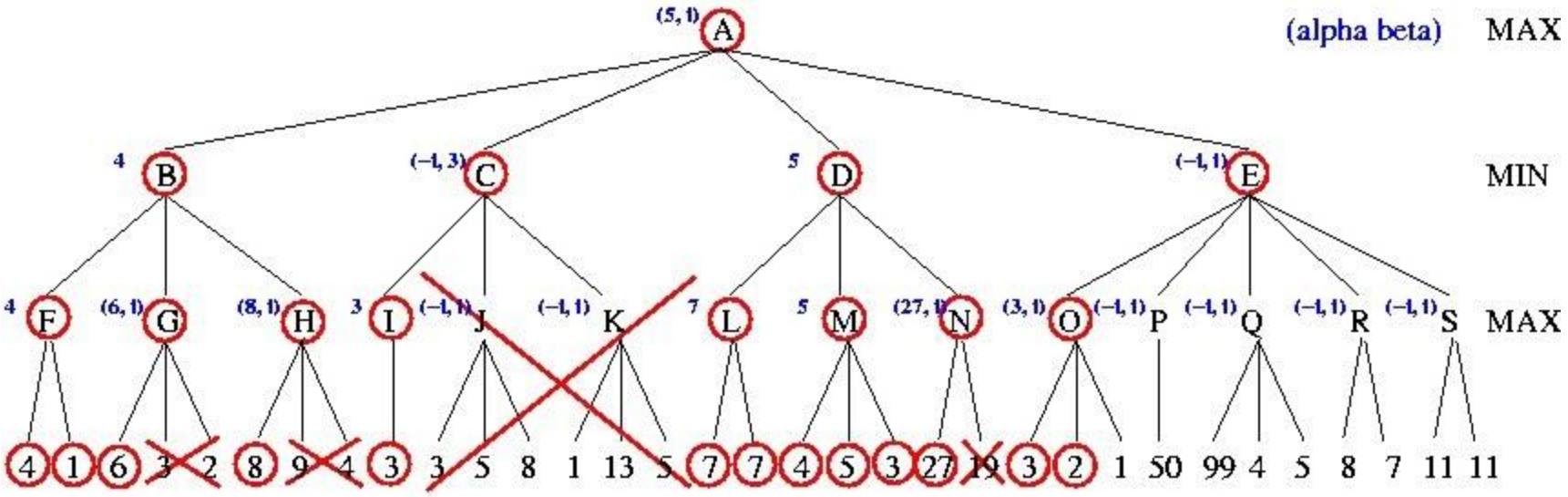
Example


(alpha beta)
MAX
MIN
MAX
A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P
Q
R
S
(5, 1)
4
(−1, 3)
5
(−1, 1)
4
(6, 1)
(8, 1)
3
(−1, 1)
(−1, 1)
7
5
(27, 1)
(−1, 1)
(−1, 1)
(−1, 1)
(−1, 1)
(−1, 1)
4 1 6 3 2 8 9 4 3 3 5 8 1 13 5 7 7 4 5 3 27 19 3 2 1 50 99 4 5 8 7 11 11

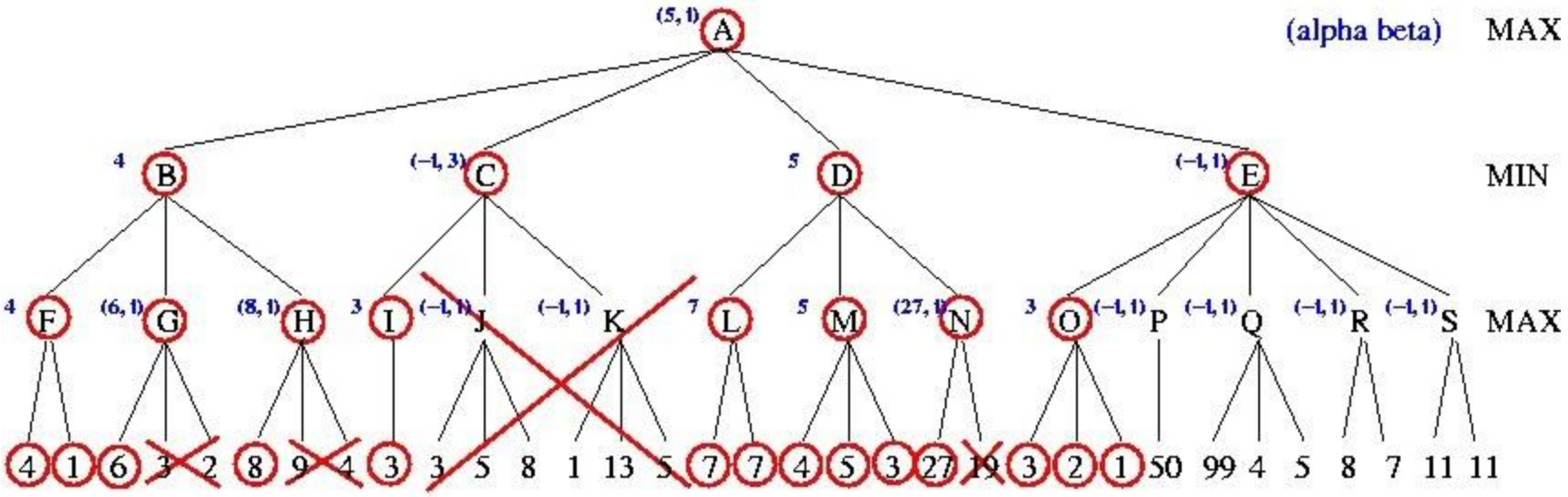
Example


(5, 1) A
(alpha beta)
MAX
4 B
(−1, 3) C
5 D
(−1, 1) E
MIN
4 F
(6, 1) G
(8, 1) H
3 I
(−1, 1) J
(−1, 1) K
7 L
5 M
(27, 1) N
(3, 1) O
(−1, 1) P
(−1, 1) Q
(−1, 1) R
(−1, 1) S
MAX
4 1 6 3 2 8 9 4 3 3 5 8 1 13 5 7 7 4 5 3 27 19 3 2 1 50 99 4 5 8 7 11 11

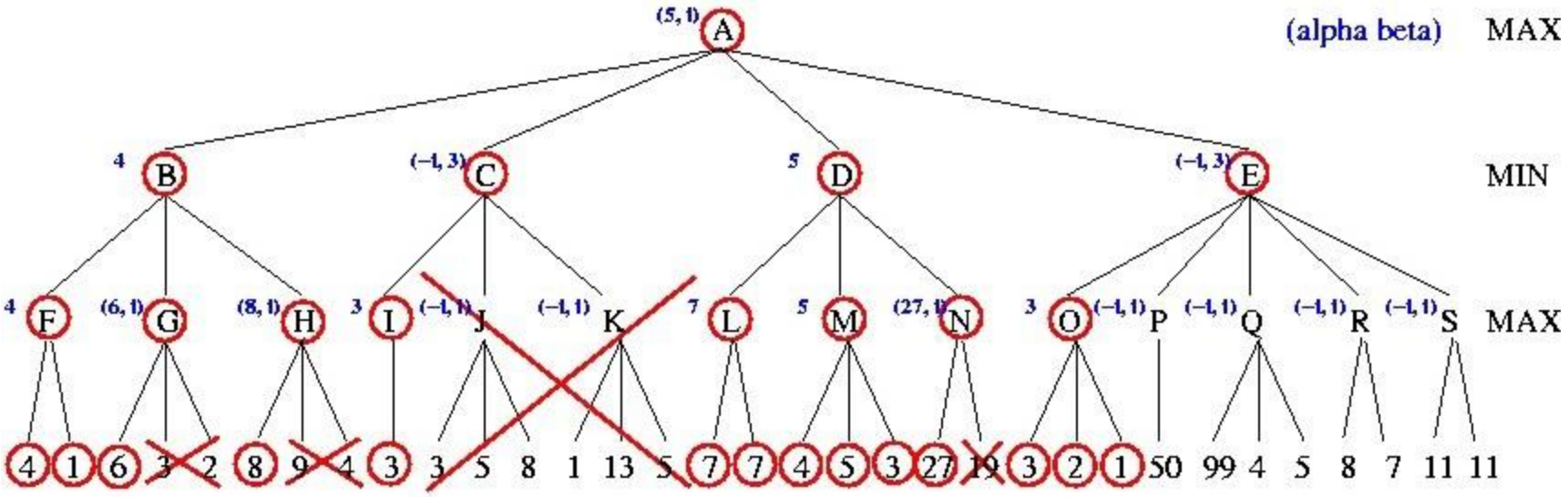
Example


(5, 1) A
(alpha beta)
MAX
4 B
(−1, 3) C
5 D
(−1, 1) E
MIN
4 F
(6, 1) G
(8, 1) H
3 I
(−1, 1) J
(−1, 1) K
7 L
5 M
(27, 1) N
(3, 1) O
(−1, 1) P
(−1, 1) Q
(−1, 1) R
(−1, 1) S
MAX
4 1 6 3 2 8 9 4 3 3 5 8 1 13 5 7 7 4 5 3 27 19 3 2 1 50 99 4 5 8 7 11 11

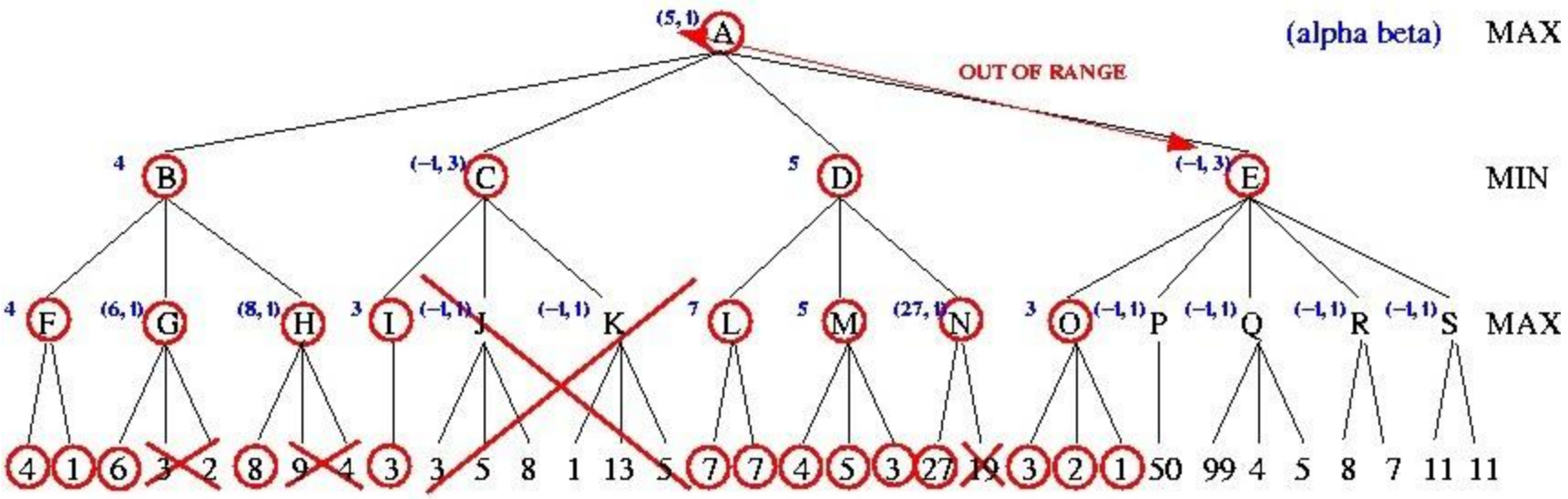
Example


(alpha beta)
MAX
MIN
MAX
(5, 1) A
4 B
(−1, 3) C
5 D
(−1, 1) E
4 F
(6, 1) G
(8, 1) H
3 I
(−1, 1) J
(−1, 1) K
7 L
5 M
(27, 1) N
3 O
(−1, 1) P
(−1, 1) Q
(−1, 1) R
(−1, 1) S
4 1 6 3 2 8 9 4 3 3 5 8 1 13 5 7 7 4 5 3 27 19 3 2 1 50 99 4 5 8 7 11 11

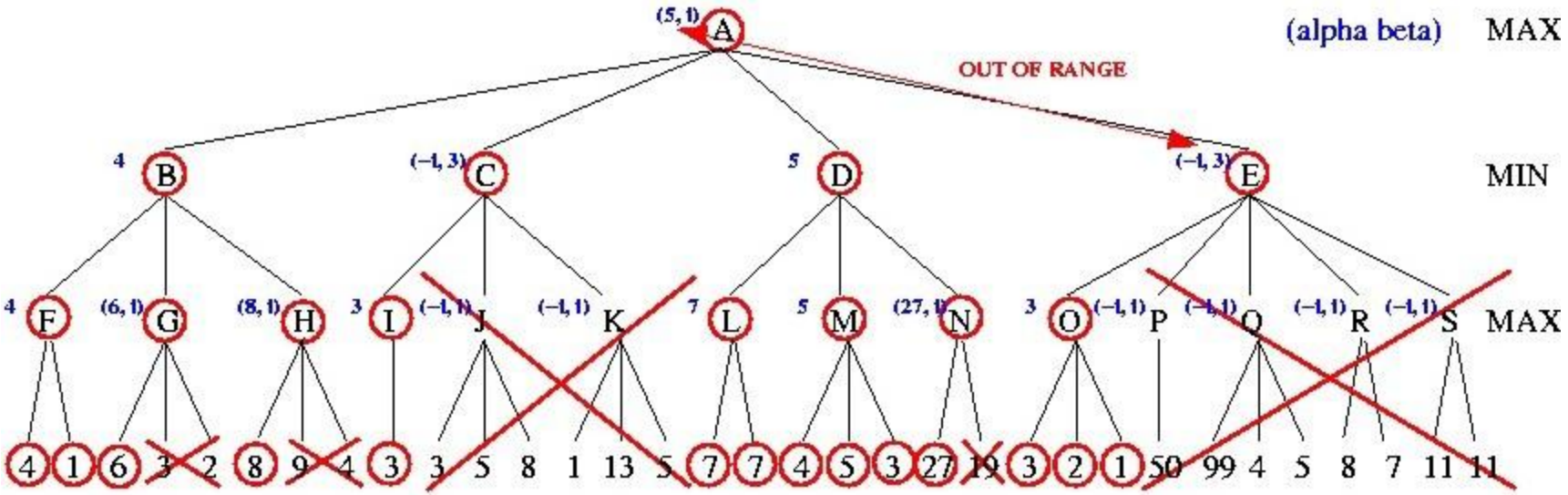
Example


(5, 1) A
(alpha beta)
MAX
4 B
(−1, 3) C
5 D
(−1, 3) E
MIN
4 F
(6, 1) G
(8, 1) H
3 I
(−1, 1) J
(−1, 1) K
7 L
5 M
(27, 1) N
3 O
(−1, 1) P
(−1, 1) Q
(−1, 1) R
(−1, 1) S
MAX
4 1 6 3 2 8 9 4 3 3 5 8 1 13 5 7 7 4 5 3 27 19 3 2 1 50 99 4 5 8 7 11 11

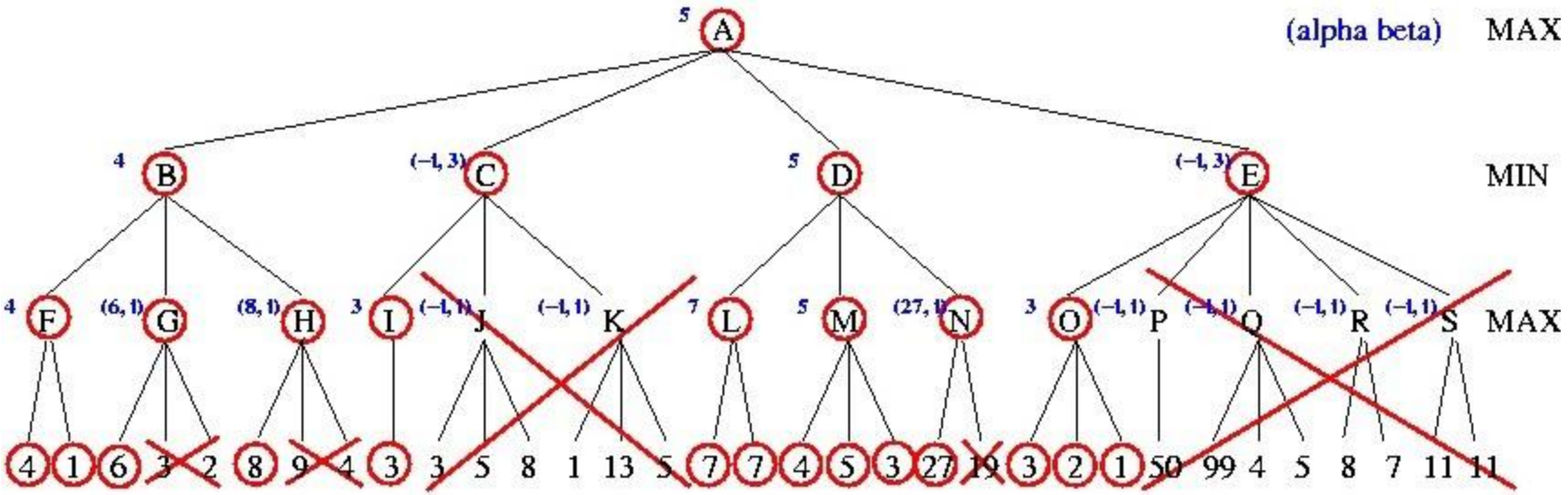
Example


(alpha beta)
MAX
MIN
MAX
OUT OF RANGE
(5, 1)
A
4
B
(−1, 3)
C
5
D
(−1, 3)
E
4
F
(6, 1)
G
(8, 1)
H
3
I
(−1, 1)
J
(−1, 1)
K
7
L
5
M
(27, 1)
N
3
O
(−1, 1)
P
(−1, 1)
Q
(−1, 1)
R
(−1, 1)
S
4 1 6 3 2 8 9 4 3 3 5 8 1 13 5 7 7 4 5 3 27 19 3 2 1 50 99 4 5 8 7 11 11

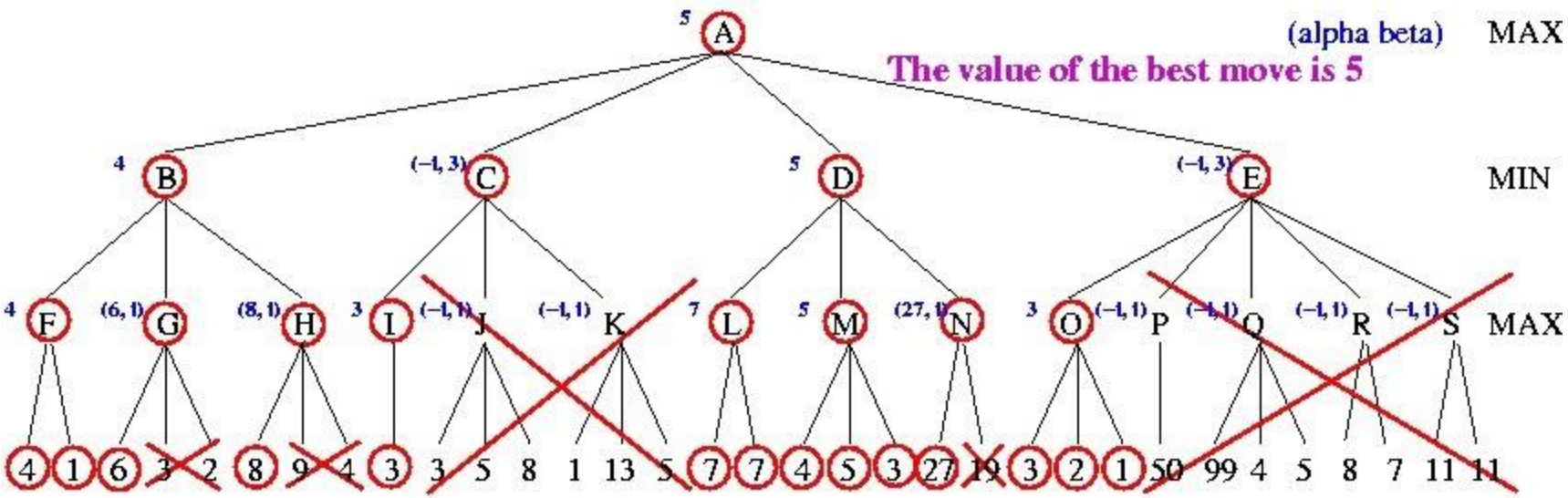
Example


(alpha beta)
MAX
MIN
MAX
OUT OF RANGE
(5, 1)
A
4
B
(−1, 3)
C
5
D
(−1, 3)
E
4
F
(6, 1)
G
(8, 1)
H
3
I
(−1, 1)
J
(−1, 1)
K
7
L
5
M
(27, 1)
N
3
O
(−1, 1)
P
(−1, 1)
Q
(−1, 1)
R
(−1, 1)
S
4 1 6 3 2 8 9 4 3 3 5 8 1 13 5 7 7 4 5 3 27 19 3 2 1 50 99 4 5 8 7 11 11

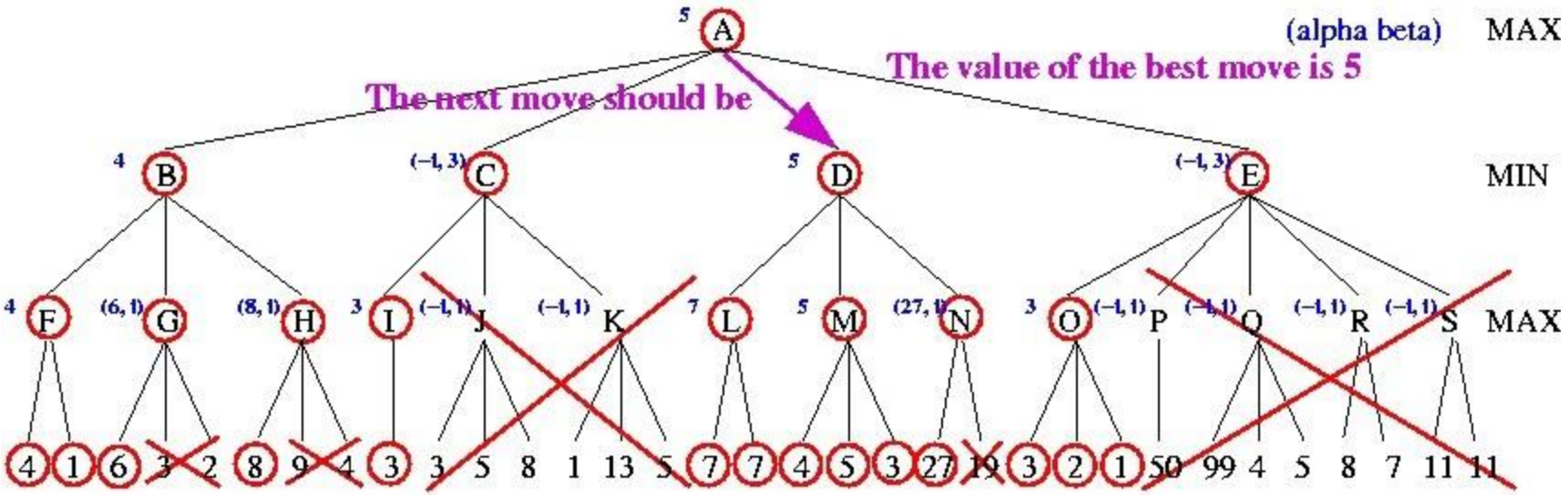
Example


5 A
(alpha beta)
MAX
4 B
(−1, 3) C
5 D
(−1, 3) E
MIN
4 F
(6, 1) G
(8, 1) H
3 I
(−1, 1) J
(−1, 1) K
7 L
5 M
(27, 1) N
3 O
(−1, 1) P
(−1, 1) Q
(−1, 1) R
(−1, 1) S
MAX
4 1 6 3 2 8 9 4 3 3 5 8 1 13 5 7 7 4 5 3 27 19 3 2 1 50 99 4 5 8 7 11 11

Example


5
A
(alpha beta)
MAX
The value of the best move is 5
4
B
(−1, 3)
C
5
D
(−1, 3)
E
MIN
4
F
(6, 1)
G
(8, 1)
H
3
I
(−1, 1)
J
(−1, 1)
K
7
L
5
M
(27, 1)
N
3
O
(−1, 1)
P
(−1, 1)
Q
(−1, 1)
R
(−1, 1)
S
MAX
4 1 6 3 2 8 9 4 3 3 5 8 1 13 5 7 7 4 5 3 27 19 3 2 1 50 99 4 5 8 7 11 11

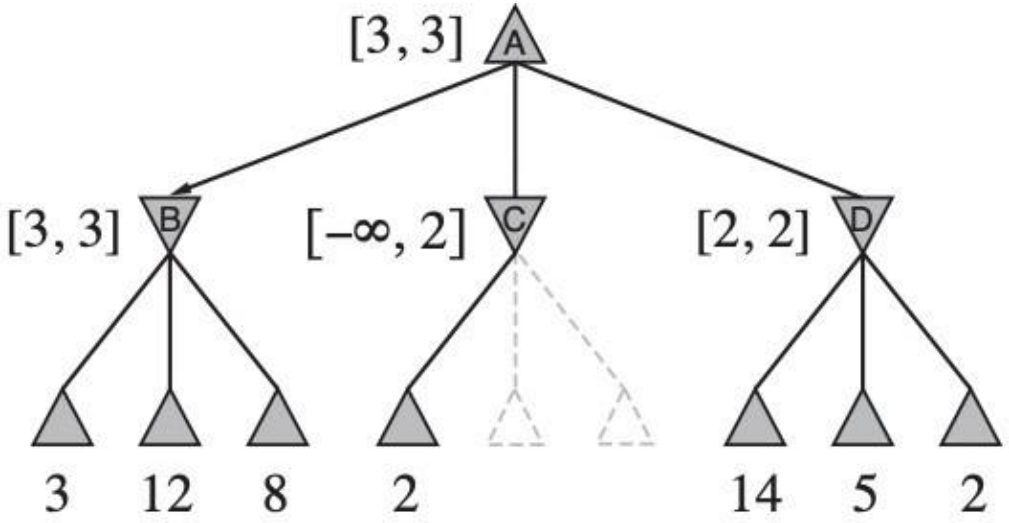
Example


5 A
(alpha beta)
MAX
The value of the best move is 5
The next move should be
4 B
(−1, 3) C
5 D
(−1, 3) E
MIN
4 F
(6, 1) G
(8, 1) H
3 I
(−1, 1) J
(−1, 1) K
7 L
5 M
(27, 1) N
3 O
(−1, 1) P
(−1, 1) Q
(−1, 1) R
(−1, 1) S
MAX
4 1 6 3 2 8 9 4 3 3 5 8 1 13 5 7 7 4 5 3 27 19 3 2 1 50 99 4 5 8 7 11 11

Alpha – Beta Pruning

Move Ordering

- The effectiveness of alpha–beta pruning is highly dependent on the order in which the states are examined.



If the third successor of D had been generated first, we would have been able to prune the other two.

Move Ordering

- If we choose a random successor, need to examine $O(b^{3m/4})$ nodes
- Choosing the best successor first, need to examine $O(b^{m/2})$ nodes
  - Effective Branching factor becomes $\sqrt{b}$ instead of b
- e.g.: For Chess, a simple ordering function is:
  1. Try captures first
  2. Then threats
  3. Then forward moves
  4. Then backward moves

  It gets us within a factor of 2 of the best-case $O(b^{m/2})$ result.

Move Ordering

- Dynamic move-ordering schemes also help
  - Try first the best moves of the past *(possibly same threats remain)*
  - Iterative deepening search
- Iterative deepening search
  - First, search 1 ply deep and record the best path of moves.
  - Then search 1 ply deeper, but use the recorded path to inform move ordering.
- The best moves are often called **killer moves** and to try them first is called the killer move heuristic.

Repeated states

- As in search, multiple sequences of moves may lead to the same state
- Again, can keep track of previously seen states (usually called a transposition table in this context)
- Transposition table is a hash table to store the evaluation of a state.
- transposition table can have a dramatic effect, sometimes as much as doubling the reachable search depth in chess.
- May not want to keep track of **all** previously seen states.
  - Some are discarded.

Use of Heuristics

- Though Alpha-beta pruning prunes a large part of the tree, we still need to search till the terminal states.
- Instead, **cut-off** the search earlier and apply a heuristic **evaluation function** to evaluate non-terminal states.
- This results in following heuristic minimax

$$\begin{cases} \text{EVAL}(s) & \text{if CUTOFF-TEST}(s,a) \\ \max_{a \in Actions(s)} \text{H-MINIMAX}(\text{RESULT}(s,a), d+1) & \text{if PLAYER}(s) = \text{MAX} \\ \min_{a \in Actions(s)} \text{H-MINIMAX}(\text{RESULT}(s,a), d+1) & \text{if PLAYER}(s) = \text{MIN}. \end{cases}$$

Evaluation Function

- An evaluation function returns an *estimate* of the expected utility of the game from a given position

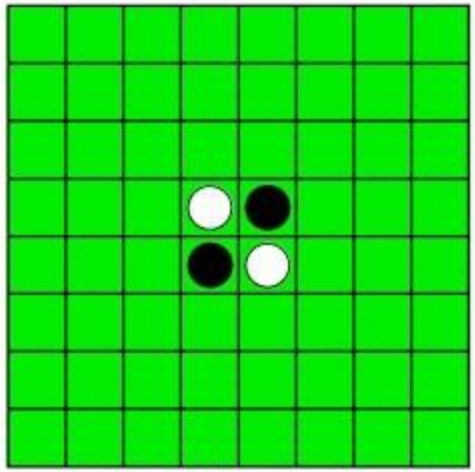
Criteria for evaluation function:

1. It should order the *terminal* states in the same way as the true utility function
2. The computation must not take too long
3. for non-terminal states, the evaluation function should be strongly correlated with the actual chances of winning.

Static Board Evaluator

- We cannot look all the way to the end of the game
  - Look ahead ply moves
  - Evaluate nodes there using SBE
- Tic Tac Toe example
  - #unblocked lines with Xs - #unblocked lines with Os
- E.g., Chess:
  - **Material value:** queen worth 9 points, rook 5, bishop 3, knight 3, pawn 1
  - Heuristic: difference between players' material values

Example



- Othello
- SBE1: #white pieces - #black pieces
- SBE2: weighted squares

Example

- **Weighted Linear function** is another heuristic
  - SBE(s) = $w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$
  - w_i is a weight and each f_i is a feature of the position
  - Nonlinear combinations of features can also be designed,
    - A pair of bishop may be more worthy than twice the value of single bishop.
- For chess:
  - 4 ply is human novice
  - 8 ply is typical PC or human master
  - 12 ply is grand master

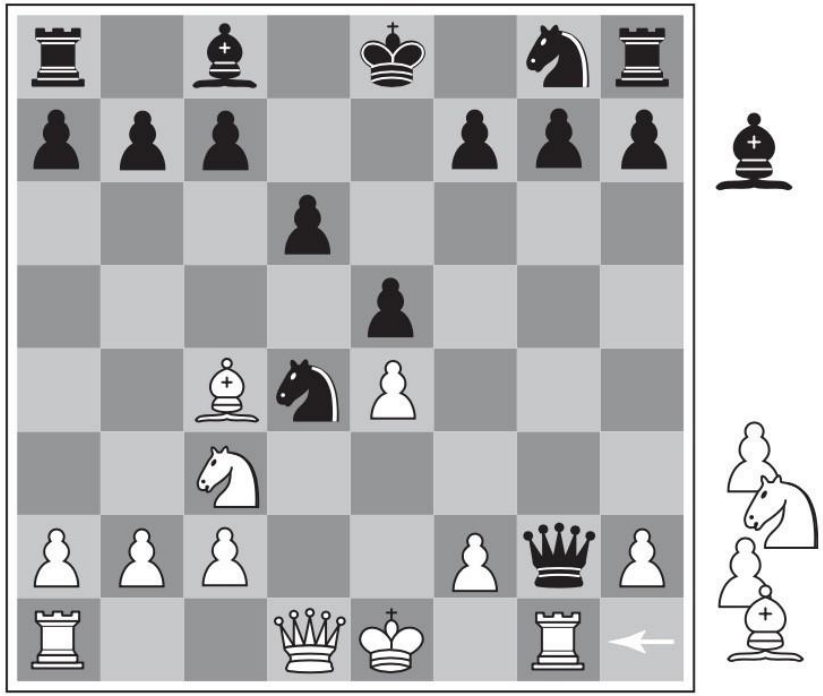
Cutting off search

Approaches for cutting-off search

- A fixed depth can be set as cut off, so that CUTOFF-TEST returns TRUE for all depths greater than d.
- Apply iterative deepening.When time runs out, return the move selected by the deepest completed search.

Errors with cutting off

- F



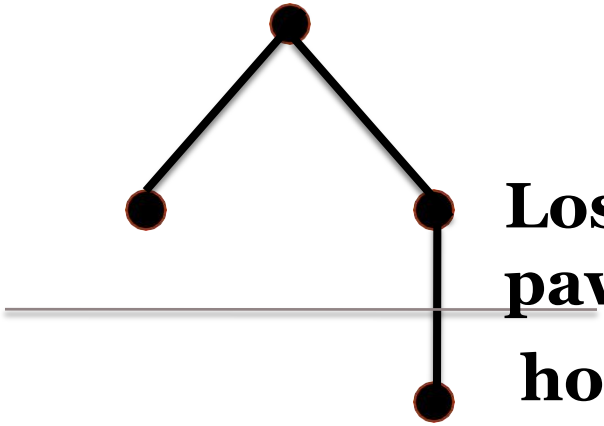
- An advantage for black, and probable win. But white's next move captures Black's queen with no compensation.

Addressing this problem

- Try to evaluate whether nodes are quiescent
  - Quiescent = evaluation function will not change rapidly in near future
  - Only apply evaluation function to quiescent nodes
- If there is an "obvious" move at a state, apply it before applying evaluation function

Problems with a fixed ply: The Horizon Effect

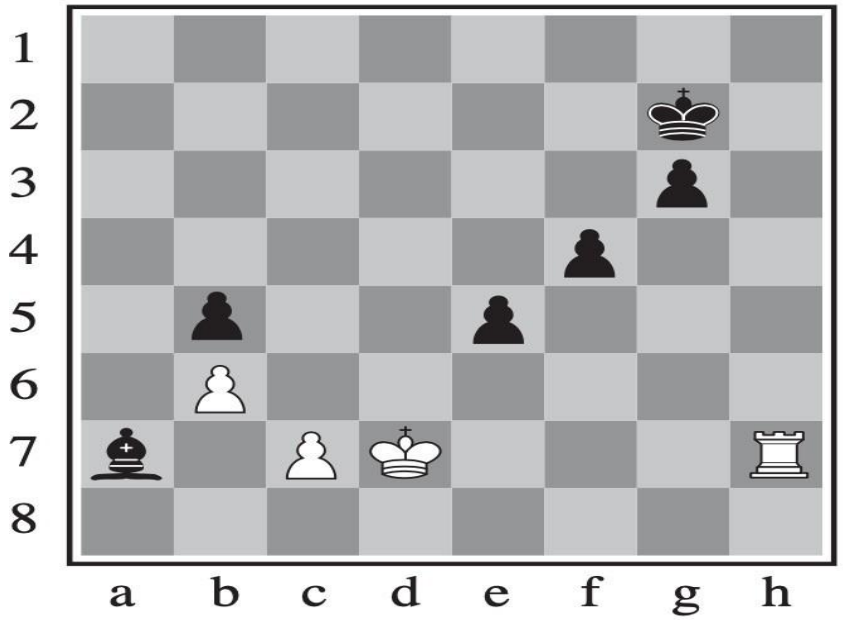
Horizon Effect: Arises when the program is facing an opponent's move that causes serious damage and is ultimately unavoidable, but can be temporarily avoided by delaying tactics.



- Inevitable losses are postponed
- Unachievable goals appear achievable
- Short-term gains mask unavoidable consequences (traps)

Horizon effect

- White Rook will capture black bishop anyhow
- Black can delay the capture by checking the king first by pawn at e5, then at f4.



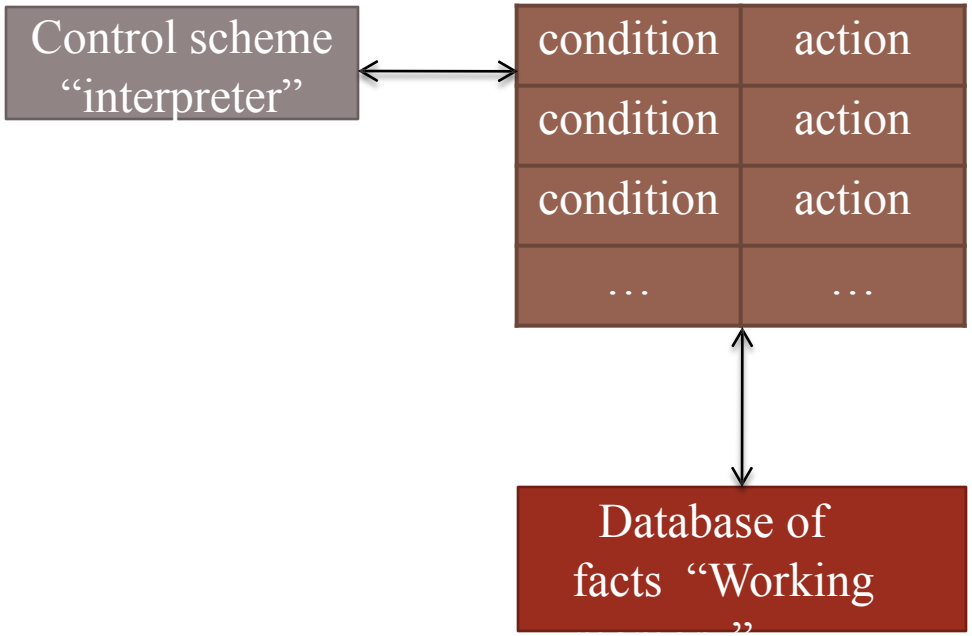
Solutions

- How to counter the horizon effect
  - Feedover
    - Do not cut off search at non-quiescent board positions (dynamic positions)
    - Example, king in danger
    - Keep searching down that path until reach quiescent (stable) nodes
  - Secondary Search
    - Search further down selected path to ensure this is the best move
  - Progressive Deepening
    - Search one ply, then two ply, etc., until run out of time
    - Similar to IDS

Expert Systems

- ES perform tasks for specific domains that require human expertise
  - Medical diagnosis, fault diagnosis, status monitoring, data interpretation, computer configuration, etc.
- ES solve problems using domain-specific knowledge
- Domain knowledge is acquired by interviewing human experts
- ES cannot operate in situations requiring common sense

Rule-Based Systems



Rule n:
IF
condition 1
condition 2
…
THEN
action 1
action 2
…

antecedent

consequent

- When one part of the IF portion matches a fact in working memory, the antecedent is SATISFIED.When all antecedents are satisfied, the rule is TRIGGERED.When the consequent of a rule is performed, the rule is FIRED.

Three Phases

1. Match phase
   - Match left side of rules (antecedents) with facts in working memory
   - Unification
2. Conflict resolution
   - Of all the rules that are triggered, decide which rule to fire.
   - Some strategies include:
     - Do not duplicate rule with same instantiated arguments twice
     - Prefer rules that refer to recently created WM elements
     - Prefer rules that are more specific (antecedents are more constraining)
     - Prefer Mammal(x) & Human(x) -> add Legs(x,2) over Legs(x,4)

       Mammal(x) -> add
   - If rules are ranked, fire according to ranking
3. Act phase
   - Add or delete facts to WM as specified by rule consequent

Expert Systems at a Glance

- ES are one of AI's early showpieces
- ES use rule-based systems along with additional capabilities
- Components of an ES include:
  - Evidence gathering
  - Control of inference
  - Uncertainty reasoning
  - Explanation generation and tutoring
  - User interface
  - Validation
  - Social issues

Some Famous Early Expert Systems

- Dendral (Stanford, Ed Feigenbaum, 1965) - organic chemical analysis
- Macsyma (MIT, 1965) - symbolic math problems
- Mycin (Stanford, 1972) - diagnose blood diseases
- Prospector (SRI, 1972) - mineral exploration
- Caduceus (1975) - internal medicine
- Xcon and R1 (1980, 1982) - computer system configuration (for DEC)
- Harpy - document retrieval
- Hearsay - Speech understanding
- Max (Nynex, 1992) - telephone network troubleshooting bacterial infection diagnosis

Examples

- Expert System Applet
- Aaron, Artist Expert System

- Some expert system shells
  - CLIPS
  - KEE
  - JESS

Mycin Example

Rules

PREMISE ($AND (SAME CNTXT GRAM GRAMNEG)
(SAME CNTXT MORH ROD)
(SAME CNTXT AIR AEROBIC))
ACTION: (CONCLUDE CNTXT CLASS ENTEROBACTERTIACEAE .8)

If the stain of the organism is granmeg, and the
morphology of the organism is rod, and the
aerobiticity of the organism is aerobic
Then there is strongly suggestive evidence (.8) that
the class of organism is enterobacteriaceae

Data
ORANISM-1:
GRAM = (GRAMNEG 1.0) MORP
= (ROD .8) (COCCUS .2)
AIR = (AEROBIC .6) (FACUL .4)

Forward Chaining and Backward Chaining

There are two main types of control schemes that are applied to rule-based systems.

Z1 If ?x has hair
Then ?x is a mammal
Z2 If ?x gives milk
Then ?x is a mammal
Z3 If ?x has feathers
Then ?x is a bird
Z6 If ?x is a mammal
?x has pointed teeth
?x has claws
?x has forward-pointing eyes
Then ?x is a carnivore
Z8 If ?x is a mammal
?x chews cud
Then ?x is an ungulate
Z11 If ?x is an ungulate
?x has long legs
?x has long neck
?x has tawny color
?x has dark spots
Then ?x is a giraffe

Database

F1) Stretch has hair
F2) Stretch chews cud
F3) Stretch has long legs
F4) Stretch has a long neck
F5) Stretch has tawny color
F6) Stretch has dark spots

Forward Chaining

- Reason FORWARD from facts/rules to (hopefully) a needed goal
- Use modus ponens to generate new facts
- Rule antecedents are compared with facts from database
- If match, add consequents to database
- Repeat as long as needed
- Forward chaining is “data driven”

Match Z1 & F1
Add: Stretch is a mammal
Match Z8/1 & F7 (?x Stretch) Match Z8/2 & F2
Add: Stretch is an ungulate
Note: Z5/1, Z6/1, Z7/1 would be matched before Z8/1

Match Z11/1 & F8 (?x Stretch) Match Z11/2 & F3
Match Z11/3 & F4 Match Z11/4 & F5 Match Z11/5 & F6
Add: Stretch is a giraffe

Backward Chaining

- Reasons BACKWARD from goal through rules to facts
- Use modus ponens
  - Start at goals
  - Match goals to consequents or facts
  - If match consequents, antecedents become new subgoals
- Repeat until
  - All subgoals are proven or
  - At least one subgoal cannot be proven
- Backward chaining is “goal driven”

Backward Chaining Example

Goal 1: Stretch is a giraffe
Match: Goal 1 and Z11/C (does not match with any facts)
Subgoal 2: Stretch is an ungulate
Subgoal 3: Stretch has long legs
Subgoal 4: Stretch has long neck
Subgoal 5: Stretch has tawny color
Subgoal 6: Stretch has dark spots
Match: Subgoal 2 and Z8/C (does not match with any facts)
Subgoal 7: Stretch is a mammal
Subgoal 8: Stretch chews cud
Match: Subgoal 7 and Z1/C (does not match with any facts)
Subgoal 9: Stretch has hair
Match: Subgoal 9 and F1
Subgoals 9, 7, met
Match: Subgoal 8 and F2
Subgoals 8, 2 met
Match: Subgoal 3 and F3
Subgoal 3 met
Match: Subgoal 4 and F4
Subgoal 4 met
Match: Subgoal 5 and F5
Subgoal 5 met
Match: Subgoal 6 and F6
Subgoal 6 met, Goal 1 met

Forward Chaining vs. Backward Chaining

- High fan out - use backward chaining
1) Human(Albert)
2) Human(Alfred)
3) Human(Barry)
4) Human(Charlie)
...
10) Human(Highlander)
...
100) Human(Shaun)
...
500) Human(Zelda)
501) Human(x) -> Mortal(x)
502) Mortal(x) -> CanDie(x) Can we kill Shaun Connery?

- FC
  503) Mortal(Albert)
  504) Moral(Alfred)
  ...
  1003) CanDie(Albert)
  1004) CanDie(Alfred)
  ...
  1100) CanDie(Shaun)
- B
  - Prove: CanDie(Shaun)
    - Match: 502/C and 1100/E
    - Prove: Mortal(Shaun)
      - Match: Mortal(Shaun) and 501/C
      - Prove: Human(Shaun)
        - Match: Human(Shaun) and 100
    - Done!

Forward Chaining vs. Backward Chaining

- Forward chaining may generate a lot of useless facts

If ?x has feathers
Than ?x can fly

If ?x has feathers
Then ?x can be used as a pen

If ?x has feathers
Then ?x is a bird
…

1 condition – many actions

Use BC!

Forward Chaining vs. Backward Chaining

- High fan in – use forward chaining
- Potential problem with BC
  - If many subgoals, each must be examined

If ?x has feathers
?x is brown
?x sings @#!@#! Patterns
?x sleeps on one foot
?x makes good pot pie
?x lives in tree
Then ?x is bird

- Many conditions – 1 action
- If time is crucial and you only have 1 goal to prove, use BC
- If you have extra time and want to be prepared for future questions, use FC to generate all possible facts
- Or use bi-directional search

Limitation of ES

- A lot of matches!
- May perform many matches for Rule 1, then have to perform same matches for Rule 2 (should share partial match information)
- May match first few antecedents, but fail on last If last antecedent added later, have to start again at beginning of rule (should save partial match information)
- One solution: RETE net (RETE stands for ``network'')
  - Contains:
    - alpha nodes (one for each antecedent)
    - beta nodes (combination of matches)
    - terminal nodes (one for each rule)
  - Explicitly store shared and partial match information
  - Machine rules and facts are numerous, matching is laborious
  - Much redundant work in match phase
  - RETE representation eliminates some redundant work

RETE Network

Rule 1

if ?x is small A1
?y is larger than A2
?x A3
?y is a carnivore

then ?y eats ?x (ADD)

Rule 2

if ?x is small
?x lives on land
?y is a carnivore
?y is strong

then ?y eats ?x (ADD)

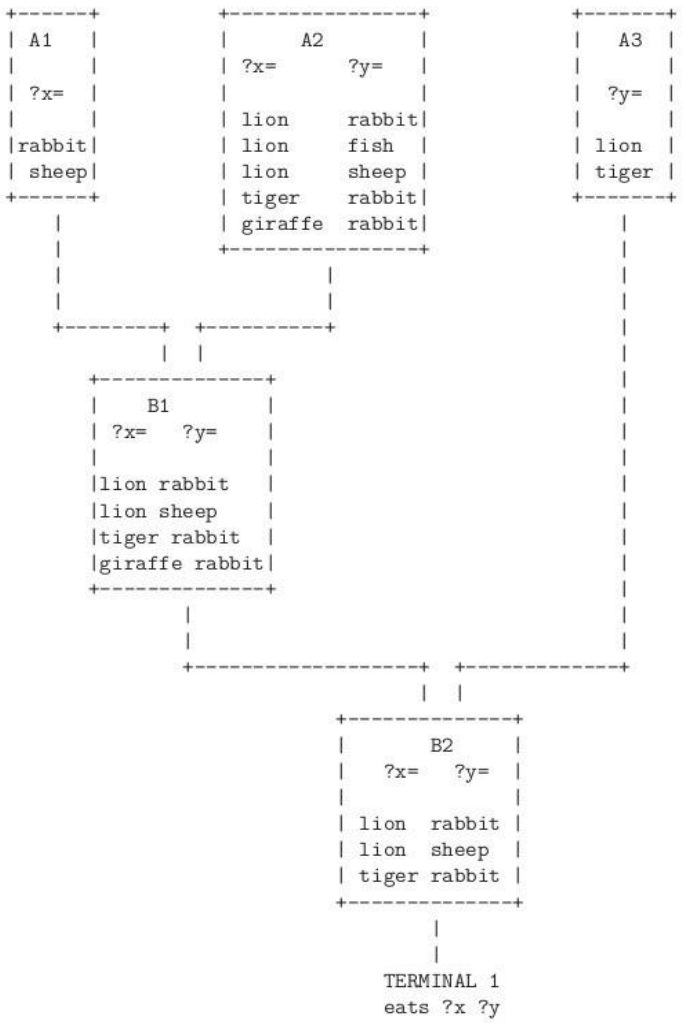
Working Memory:

rabbit is small
sheep is small
bird is small
mouse is small
lion is a carnivore
tiger is a carnivore
lion is larger than rabbit
lion is larger than fish
lion is larger than sheep
tiger is larger than rabbit
giraffe is larger than rabbit

RETE Network

- For each rule
  - For each antecedent
    - Create alpha node
    - Store antecedent matches in this node
  - For each alpha node except the first
    - Create beta node
    - If this is the first beta node in the network
      - Then parents are alpha1 and alpha2
      - Else parents are previous beta and next unmatched alpha
    - Store partial matches in this node
  - Create terminal node for each rule
    - Parent is last beta for the rule
- To run
  - Distribute assertions to top of network
  - Match assertions with alpha nodes - pass bindings to beta nodes
  - For each beta node, JOIN by checking for consistent bindings
  - When an assertion (binding list) hits terminal node, change WM
  - For each WM change, add information to alpha node and propagate down
  - Do this for additions and deletions

RETE Netw


A1
?x=
rabbit
sheep
A2
?x= ?y=
lion rabbit
lion fish
lion sheep
tiger rabbit
giraffe rabbit
A3
?y=
lion
tiger
B1
?x= ?y=
lion rabbit
lion sheep
tiger rabbit
giraffe rabbit
B2
?x= ?y=
lion rabbit
lion sheep
tiger rabbit
TERMINAL 1
eats ?x ?y