BITS Pilani
Pilani Campus

Artificial & Computational Intelligence

DSE CLZG557

M3 : Game Playing & Constraint Satisfaction

Raja vadhana P

Assistant Professor,

BITS - CSIS

Course Plan

M1	Introduction to AI
M2	Problem Solving Agent using Search
M3	Game Playing, Constraint Satisfaction Problem
M4	Knowledge Representation using Logics
M5	Probabilistic Representation and Reasoning
M6	Reasoning over time, Reinforcement Learning
M7	AI Trends and Applications, Philosophical foundations

Module 3 : Part -1

Searching to play games

A. Minimax Algorithm

B. Alpha-Beta Pruning

C. Making imperfect real time decisions

The Part C will not be a part of the mid term exam. Will be continued post mid term

Learning Objective

1. Convert a given problem into adversarial search problem
2. Formulate the problem solving agent components
3. Design static evaluation function value for a problem
4. Construct a Game tree
5. Apply Min-Max .
6. Apply and list nodes pruned by alpha pruning and nodes pruned by beta pruning

Problem Formulation

Game Problem

Study & design of games enables the computers to model ways in which humans think & act hence simulating human intelligence.

<u>AI for Gaming:</u>

- Interesting & Challenging Problem
- Larger Search Space Vs Smaller Solutions
- Explore to better the Human Computer Interaction

<u>Characteristics of Games:</u>

- Observability
- Stochasticity
- Time granularity
- Number of players

<u>Adversarial Games:</u>

Goals of agents are in conflict where one's optimized step would reduce the utility value of the other.

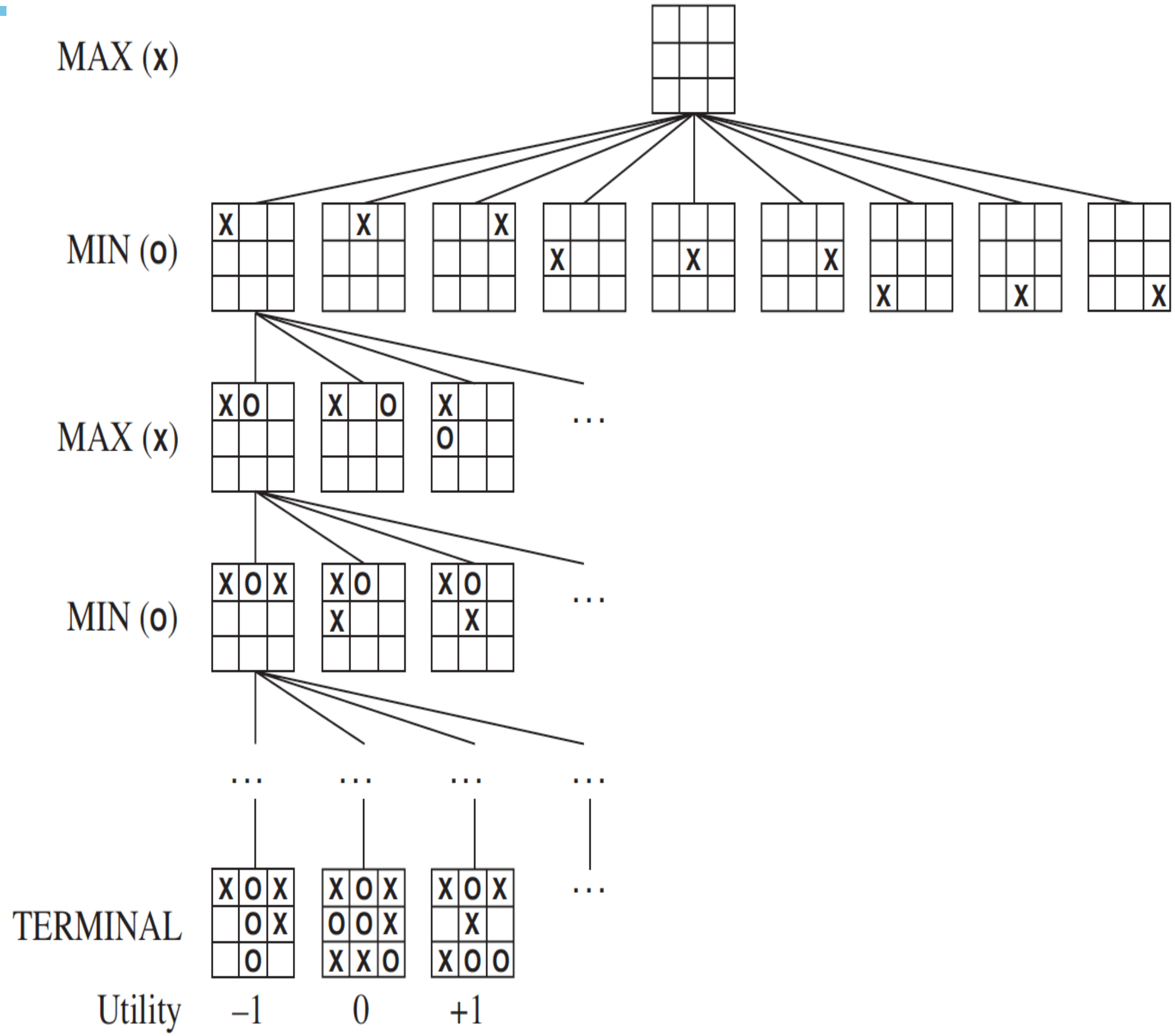
Game Problem

Games as Search Problem

PSA : Representation of Game:

INITIAL STATE: S0
PLAYER(s)
ACTIONS(s)
RESULT(s, a)
TERMINAL-TEST(s)
UTILITY(s, p)

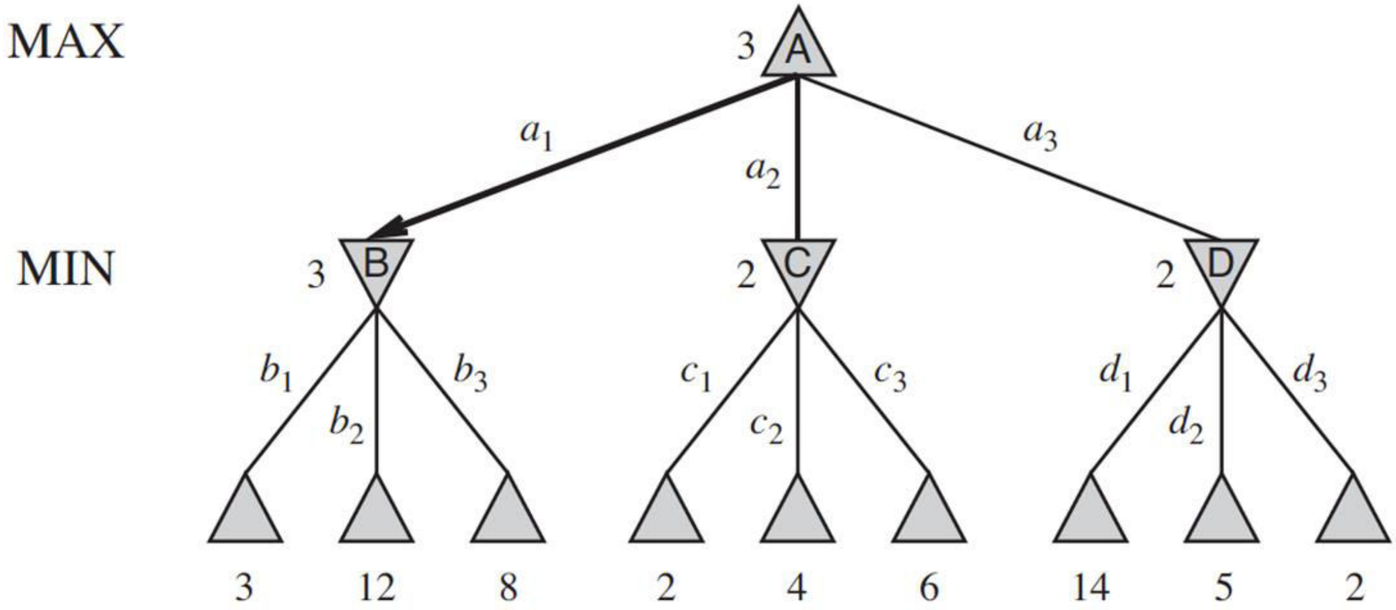
Eg., Tic Tac Toe



<u>Assumption Task Environment:</u>

Static
Strategic
Multi agent
Fully observable
Sequential
Discrete

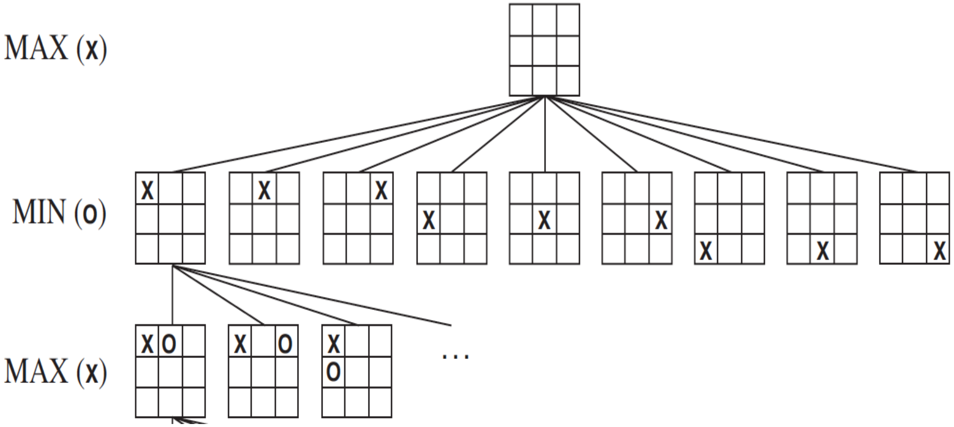
Min-Max Algorithm - Idea



Traversal on Game Tree

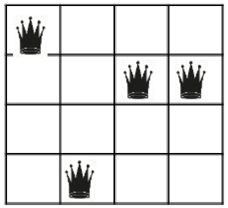
How does a player decide which action to take?

Rule Engine : Helps build a search tree superimposed on a game tree and examines enough nodes to allow a player to determine what move to make next.



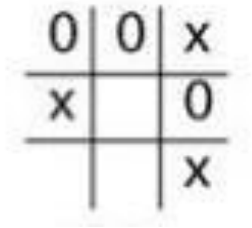
Static Evaluation Function

N-Queens



1	4	2	2	4

Tic-Tac-Toe



Max's Share	2
Min's Share	1
Board Value	1

N-Tile

2	8	3
1	6	4
7		5

1	2	3
8		4
7	6	5

No.of.Tiles Out of Place	5

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

$$= 0.6\,(MaxChance - MinChance) + 0.4\,(MaxPairs - MinPairs)$$

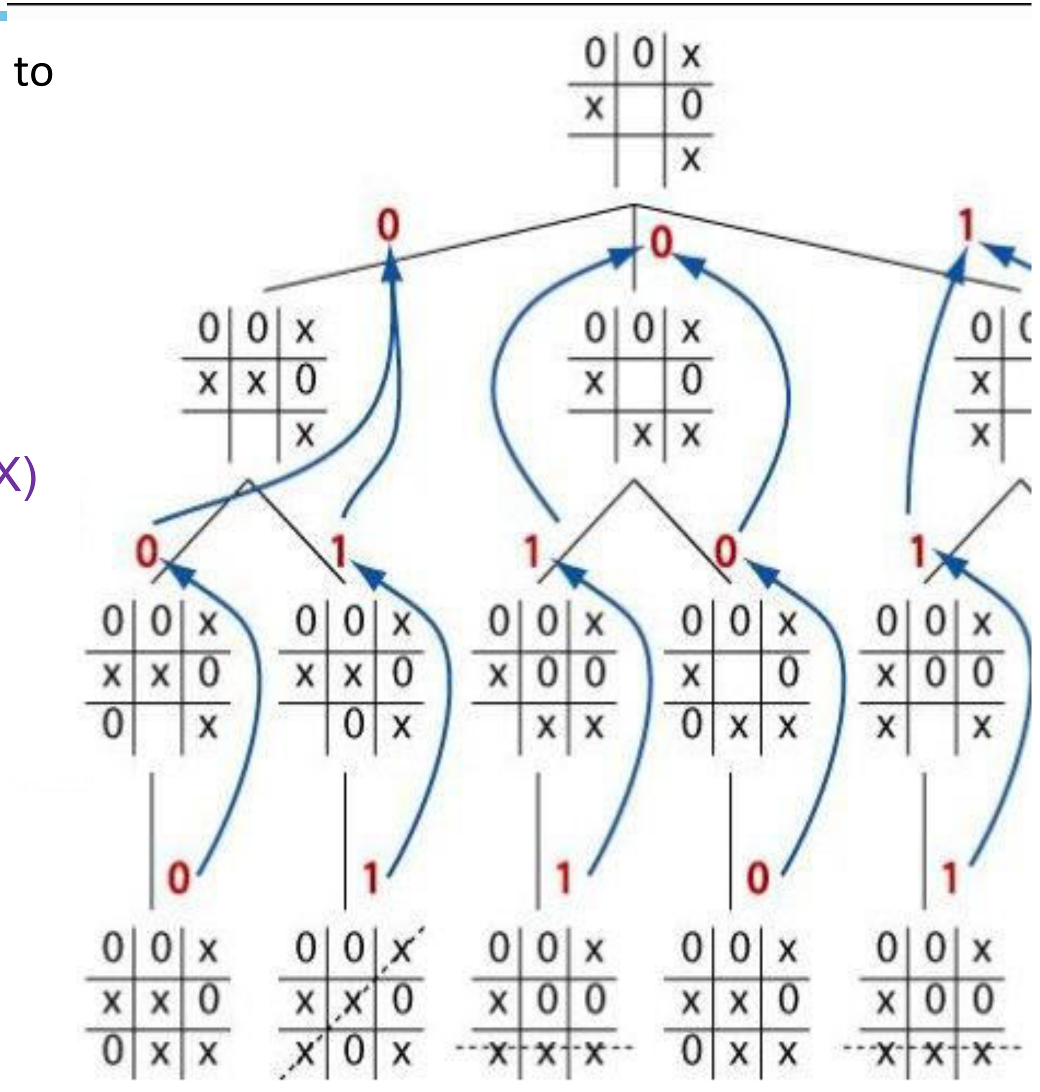
MINIMAX ALGORITHM

Min-Max Algorithm

<u>Idea:</u> Uses Depth – First search exploration to decide the move

Let
start Player = MAX
Depth m =3

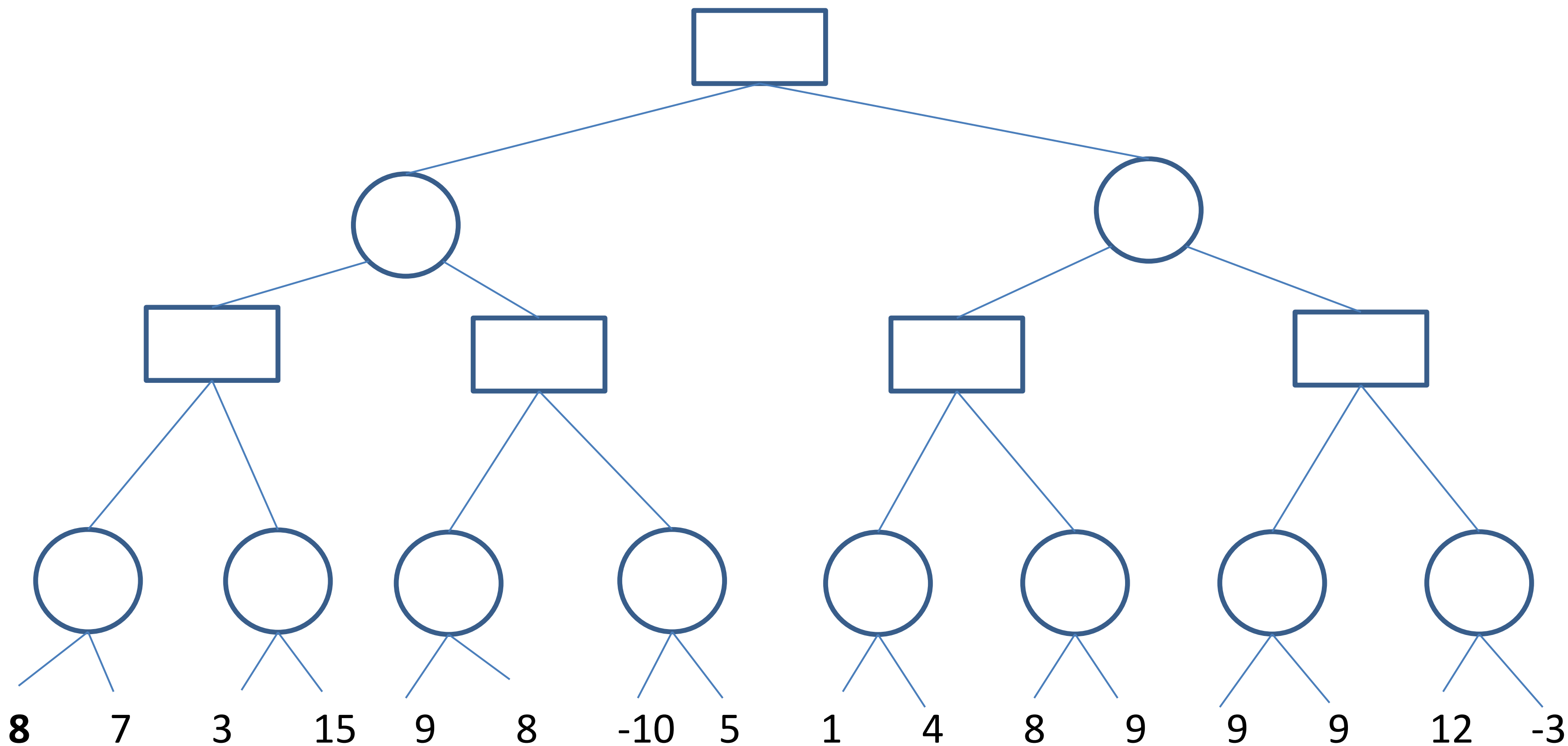
Minimax value of a node: Utility (of MAX) of being in the corresponding state, assuming both players would play optimally from the node n till the end of game

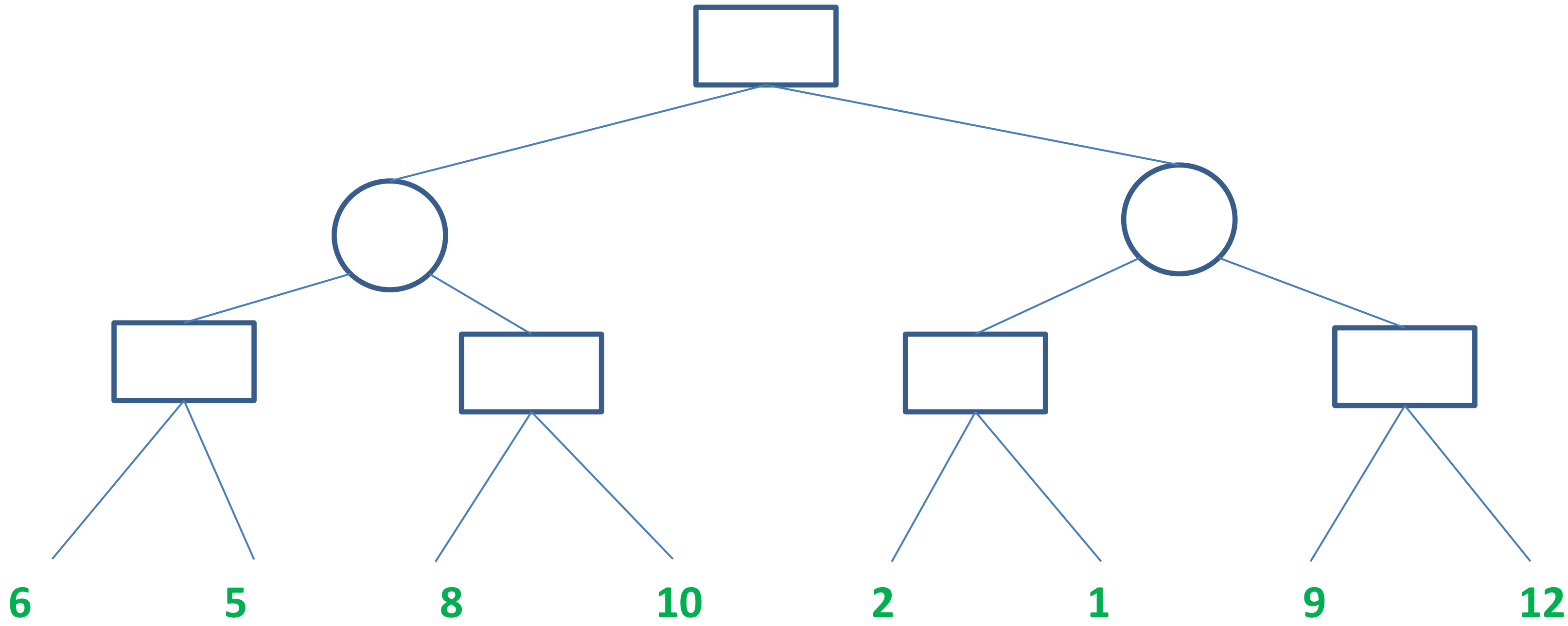


Min-Max Algorithm – Example -1

Squares represent MAX nodes

Circles represent MIN nodes

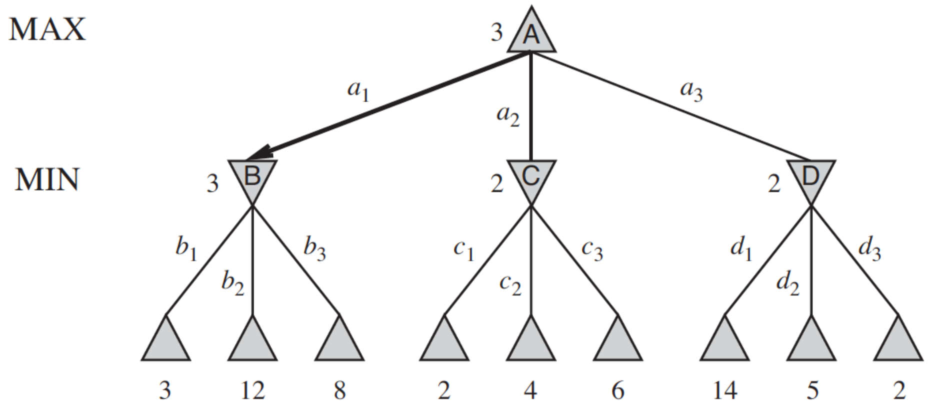



6
5
8
10
2
1
9
12

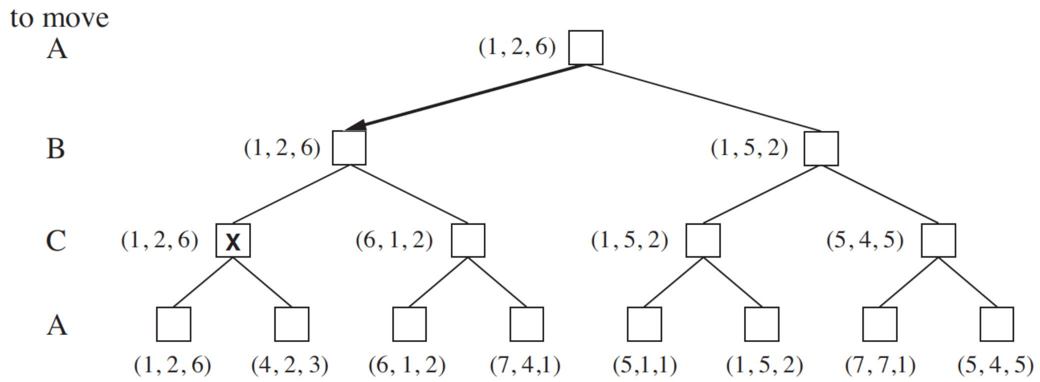
Min-Max Algorithm

```
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
```

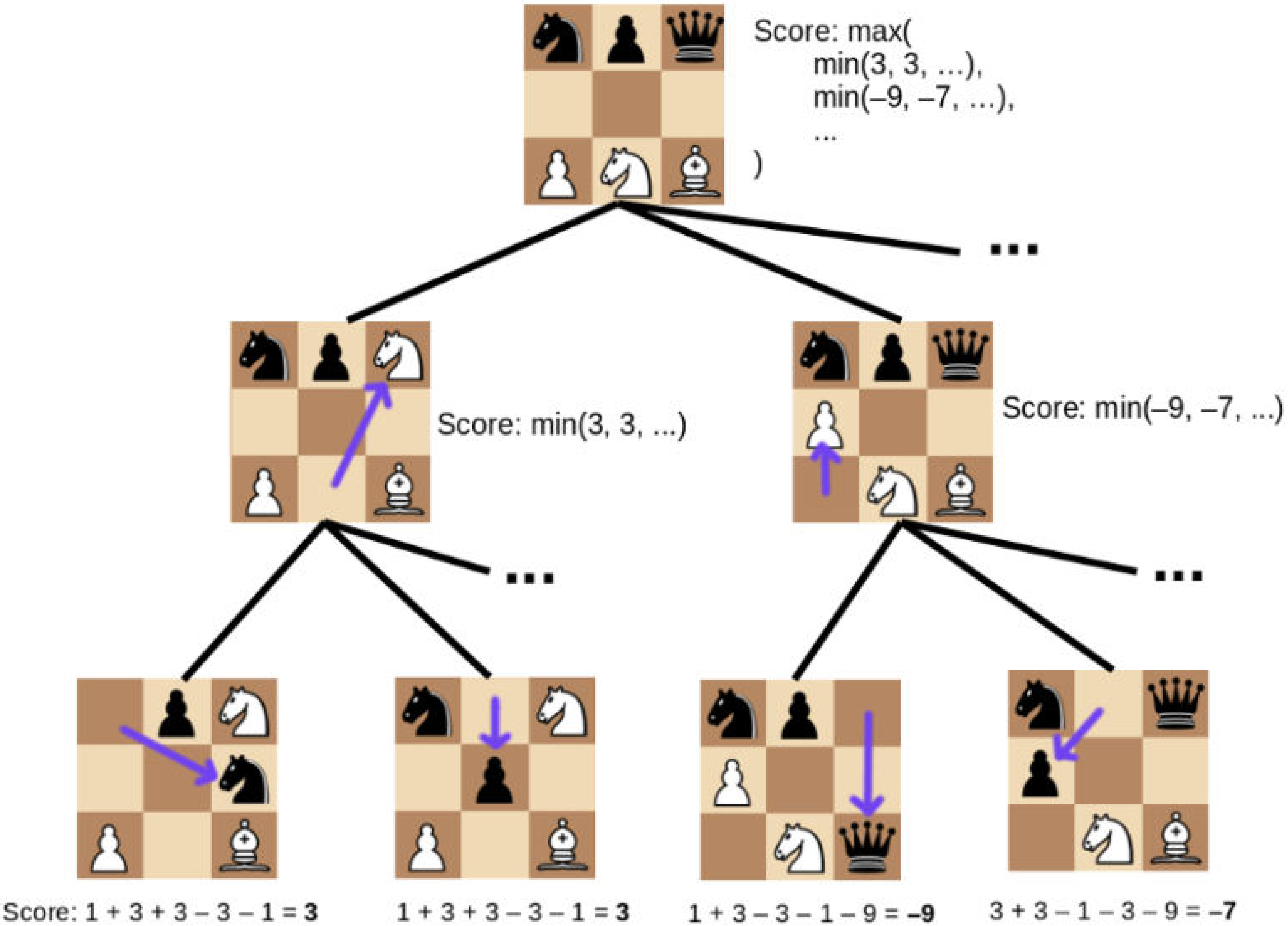
Is it possible to compute the minimax decision for a node without looking at every successor node?



Two Player Game : 1-Ply Game

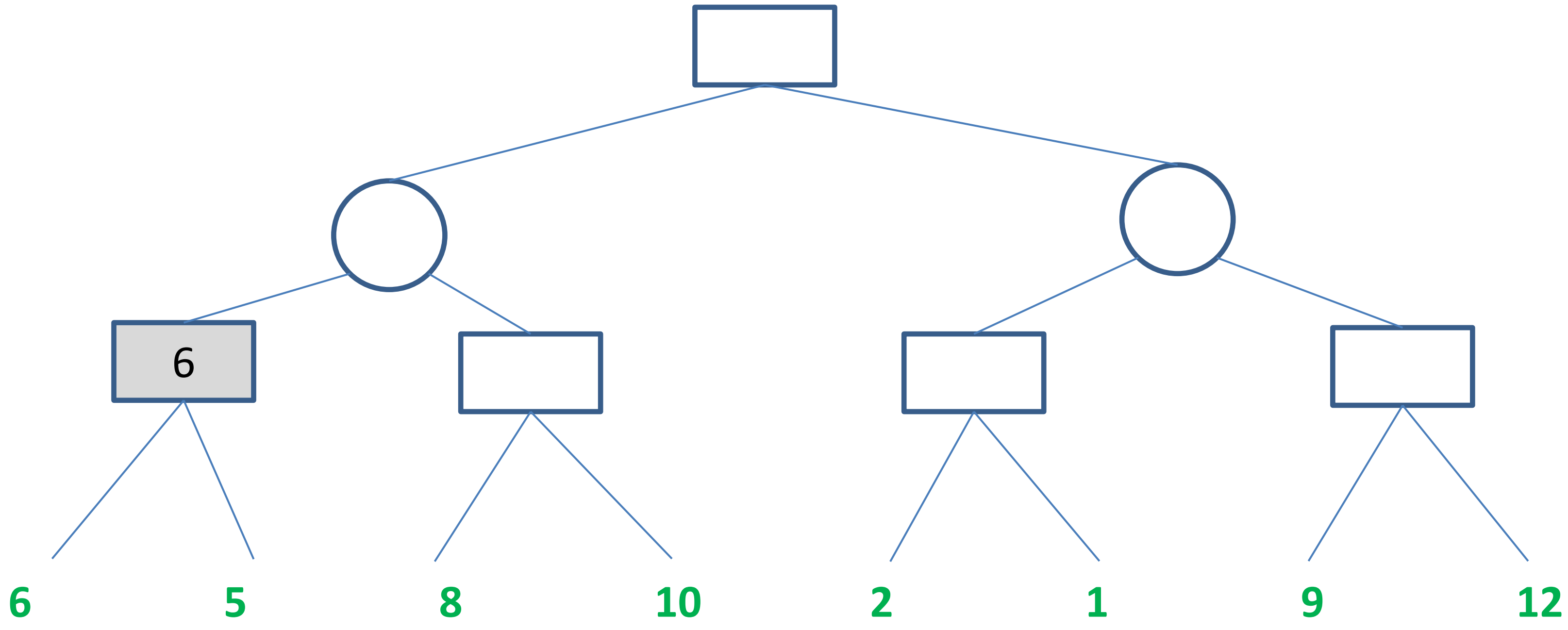


Multiplayer Game

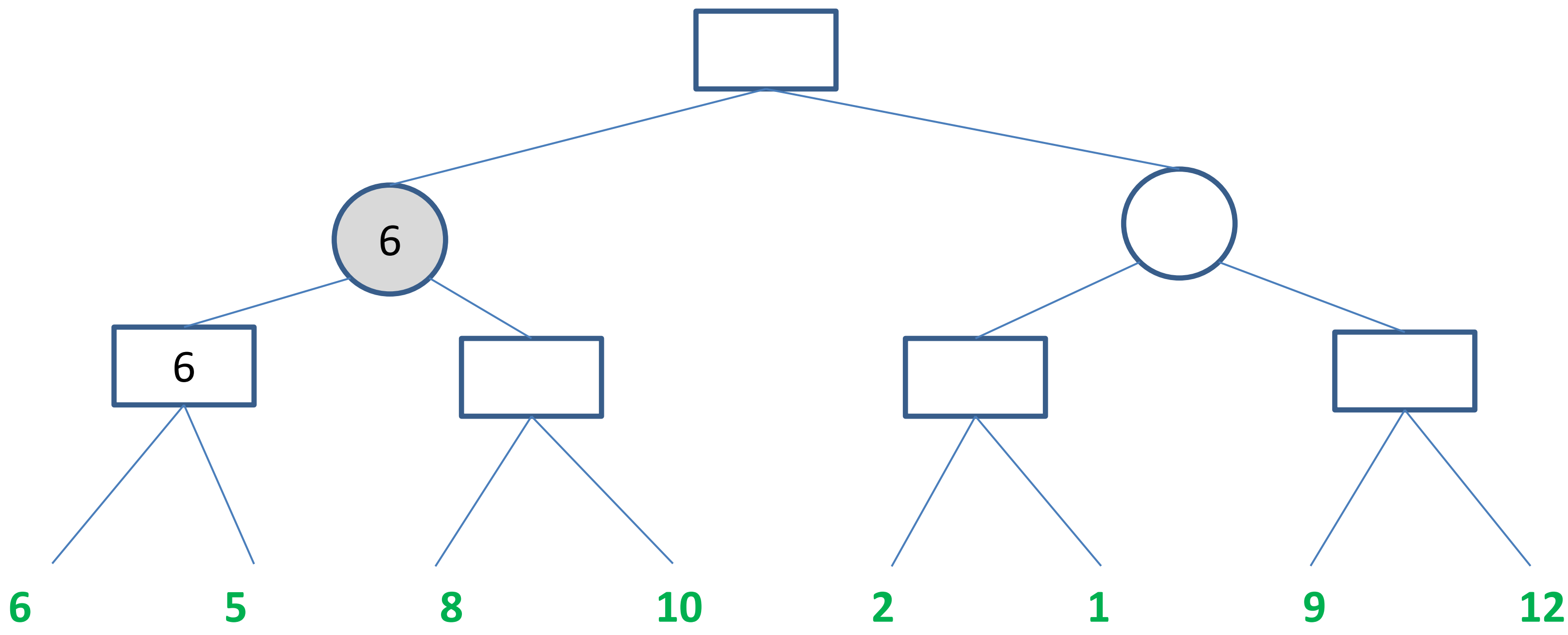

Score: max(
min(3, 3, ...),
min(–9, –7, ...),
...
)
...
Score: min(3, 3, ...)
Score: min(–9, –7, ...)
...
...
Score: 1 + 3 + 3 – 3 – 1 = 3
1 + 3 + 3 – 3 – 1 = 3
1 + 3 – 3 – 1 – 9 = –9
3 + 3 – 1 – 3 – 9 = –7

ALPHA BETA PRUNING

Idea – Beta Pruning

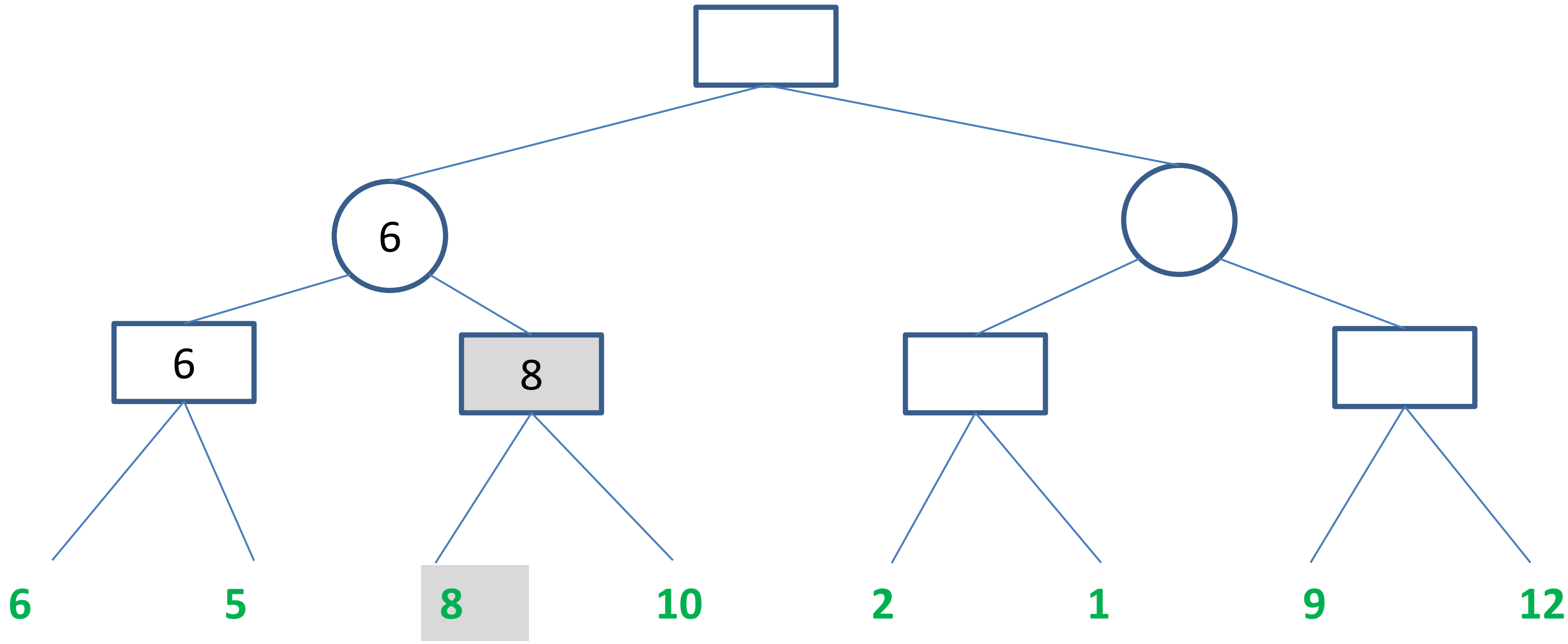


Idea – Beta Pruning

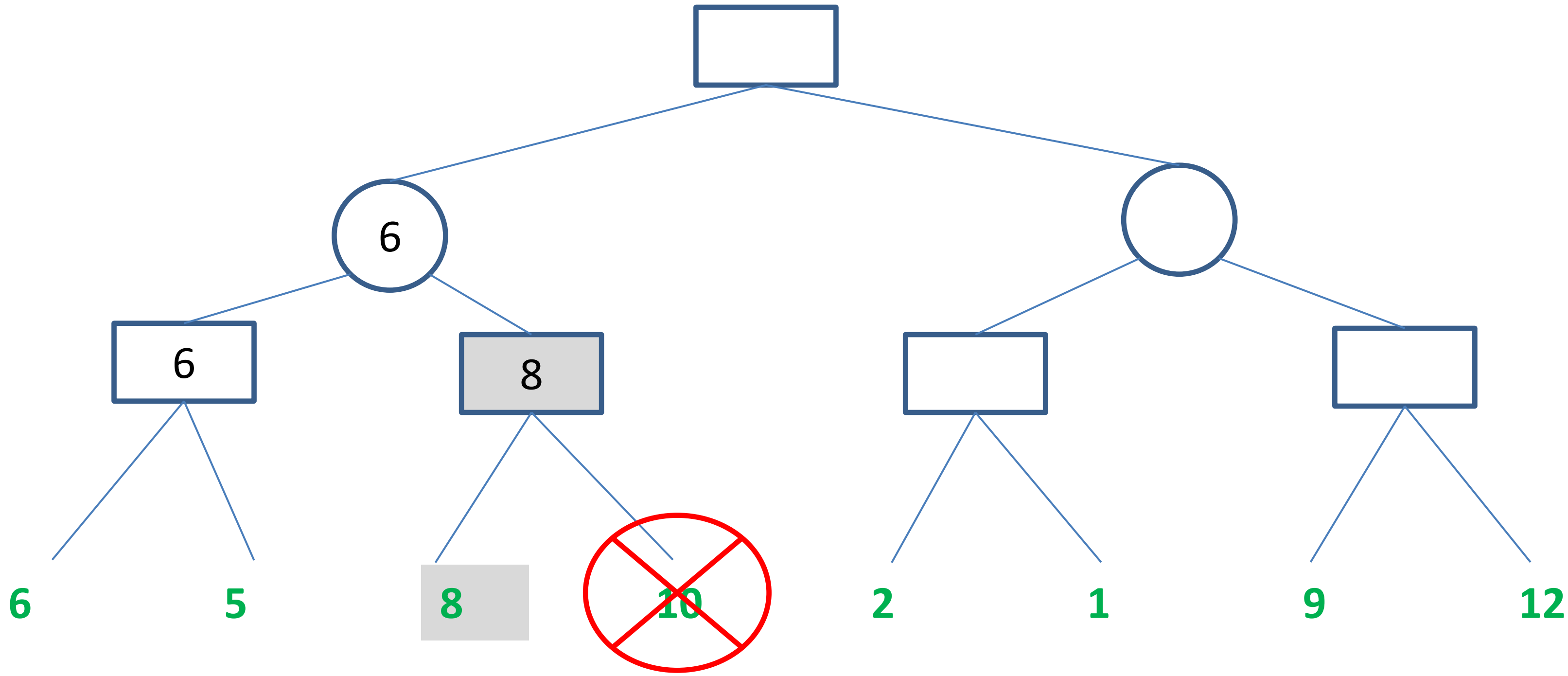


innovate achieve lead

Idea – Beta Pruning

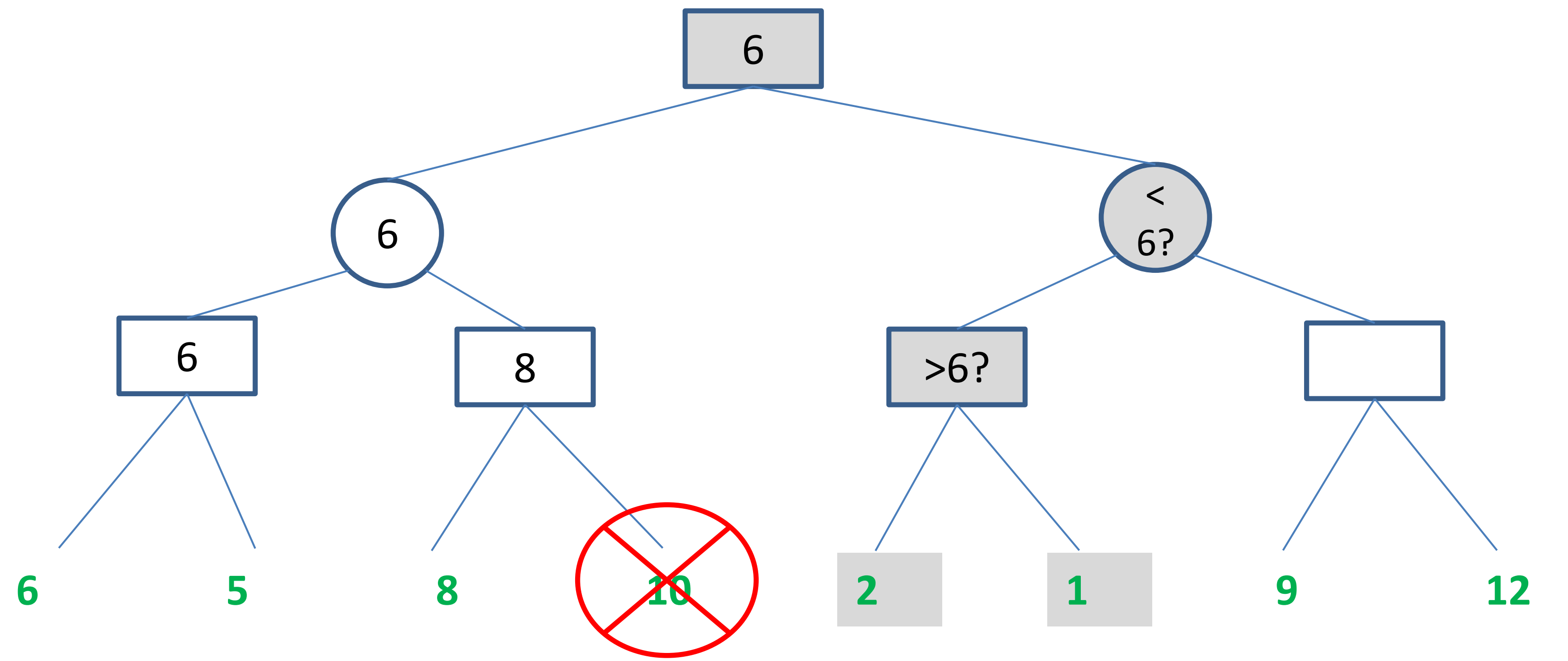


Idea – Beta Pruning

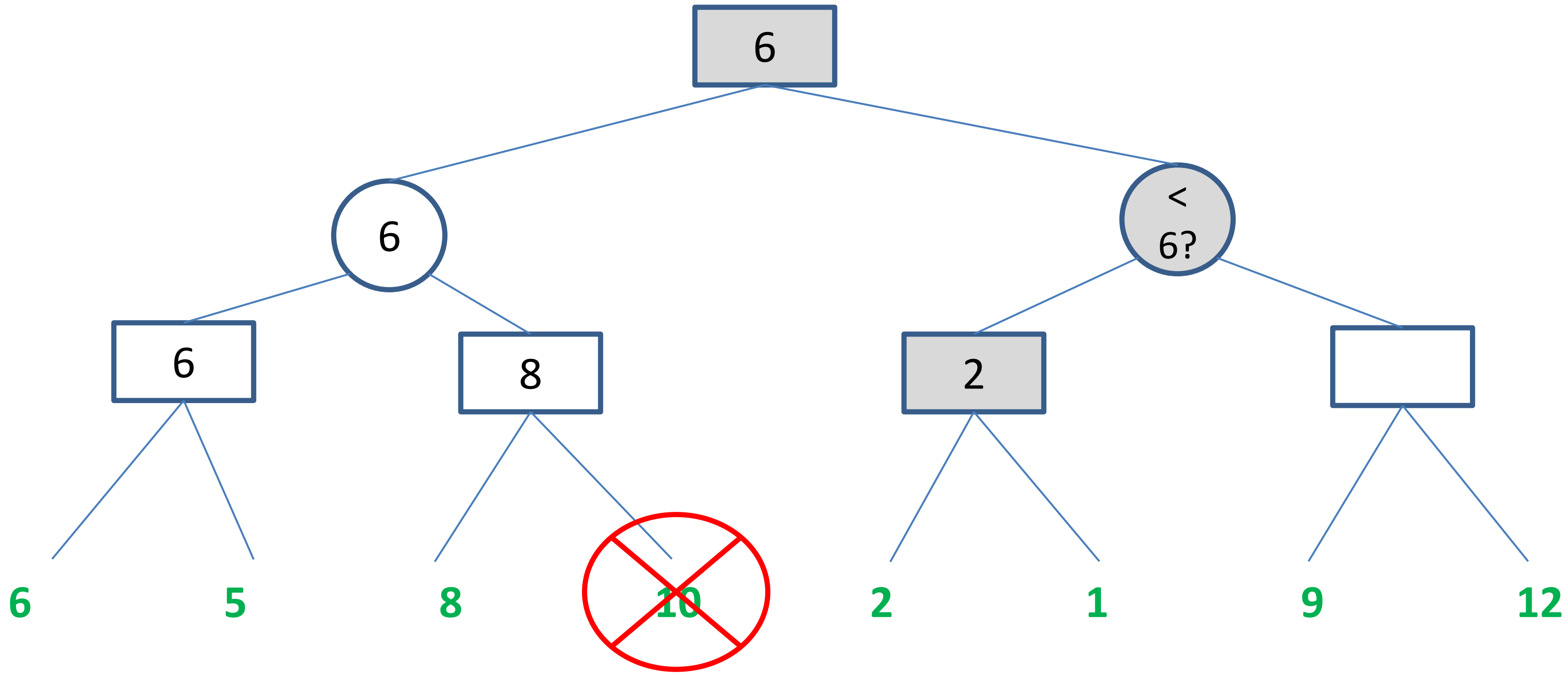


Beta – Upper Bound of Minimizer's value. Perceived value that Minimizer hopes to get against a competitive Maximizer

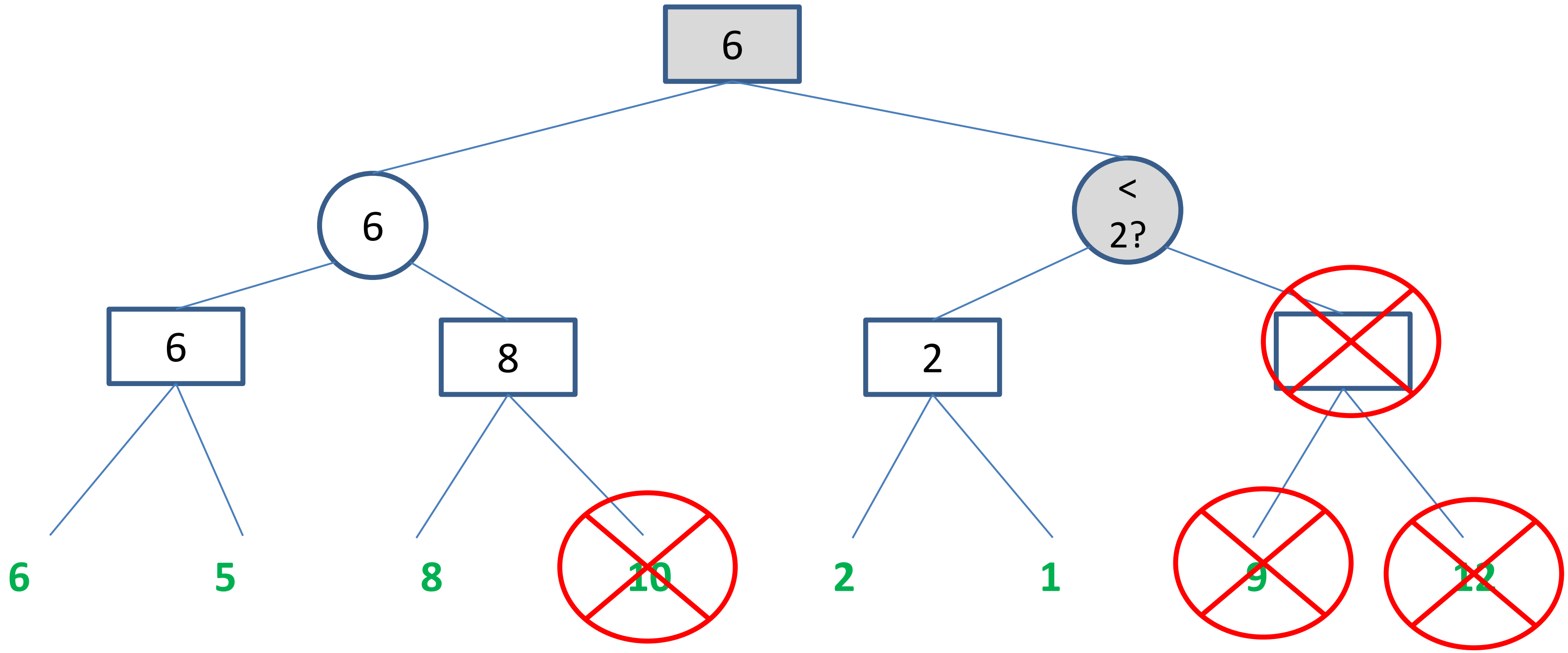
Idea – Alpha Pruning



Idea – Alpha Pruning



Idea – Alpha Pruning

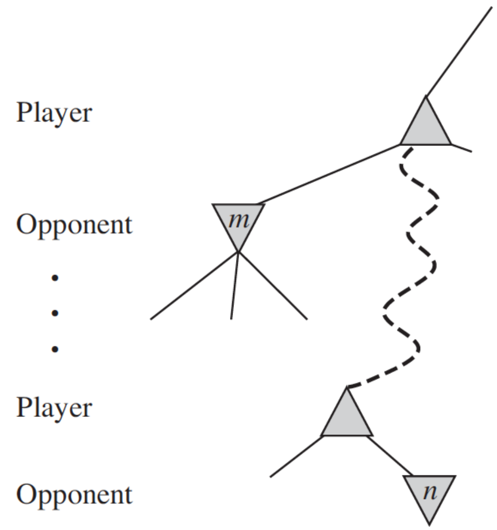


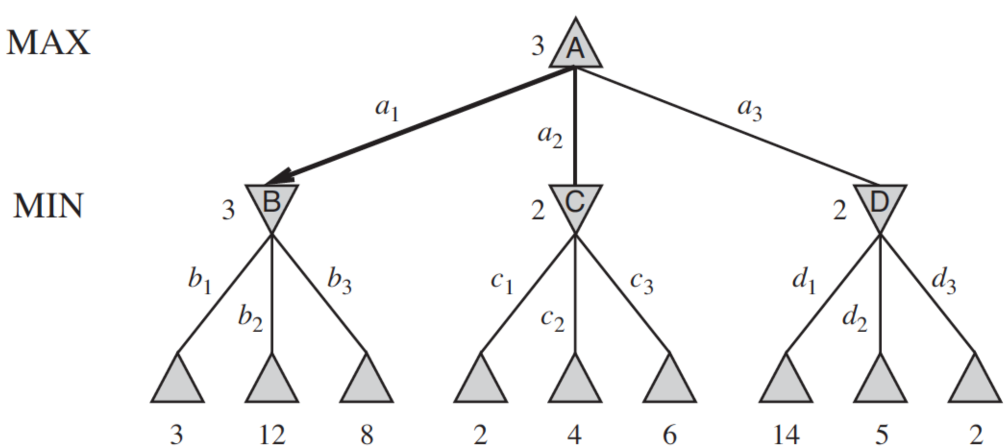
Alpha – Lower bound of Maximizer's value. Perceived value that Maximizer hopes to against a competitive Minimizer

Alpha – beta Pruning

General Principle:

At a node ***n*** if a player has better option at the parent of ***n*** or further up, then ***n*** node will never be reached .Hence the entire subtree from node ***n*** can be pruned





$$
\begin{aligned}
\textsc{Minimax}(root) &= \max(\min(3,12,8), \min(2,x,y), \min(14,5,2)) \\
&= \max(3, \min(2,x,y), 2) \\
&= \max(3, z, 2) \qquad \text{where } z = \min(2,x,y) \le 2 \\
&= 3.
\end{aligned}
$$

Alpha – beta Pruning

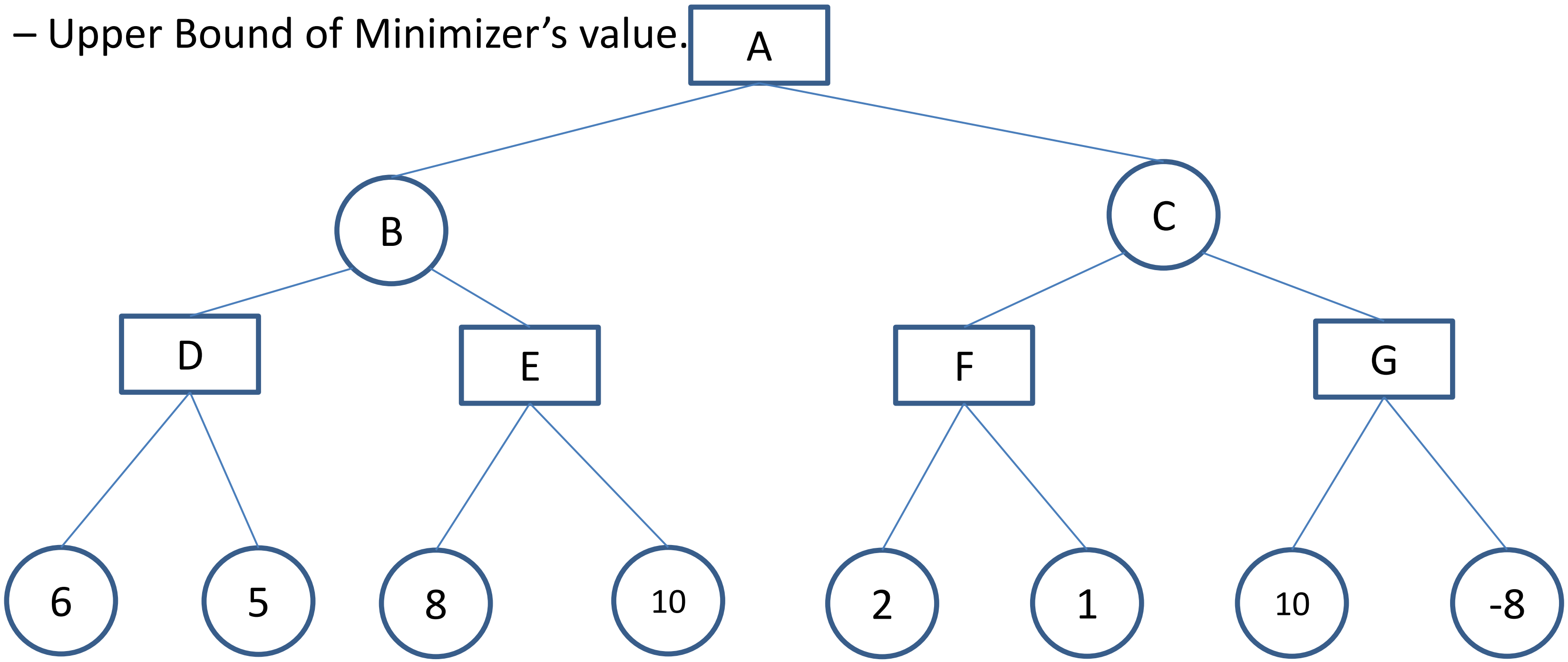
Steps in Alpha – Beta Pruning

1. At root initialize alpha = -∞ and beta = + ∞. This is to set the worst case boundary to start the algorithm which aims to increase alpha and decrease beta as much as optimally possible
2. Navigate till the depth / limit specified and get the static evaluated numeric value.
3. For every value VAL being analyzed : Loop till all the leaf/terminal/specified state level nodes are analyzed & accounted for OR until **beta <= alpha.**
   1. If the player is MAX :
      1. If VAL > alpha
      2. then reset alpha = VAL
      3. also check **if** beta <=alpha **then** tag the path as unpromising (TO BE AVOIDED) **and** prune the branch from game tree. Rest of their siblings are not considered for analysis
   2. Else if the player is MIN:
      1. If VAL < beta
      2. then reset beta = VAL
      3. also check **if** beta <=alpha **then** tag the path as unpromising (TO BE AVOIDED) **and** prune the branch from game tree. Rest of their siblings are not considered for analysis

Alpha – beta Pruning – Example -3

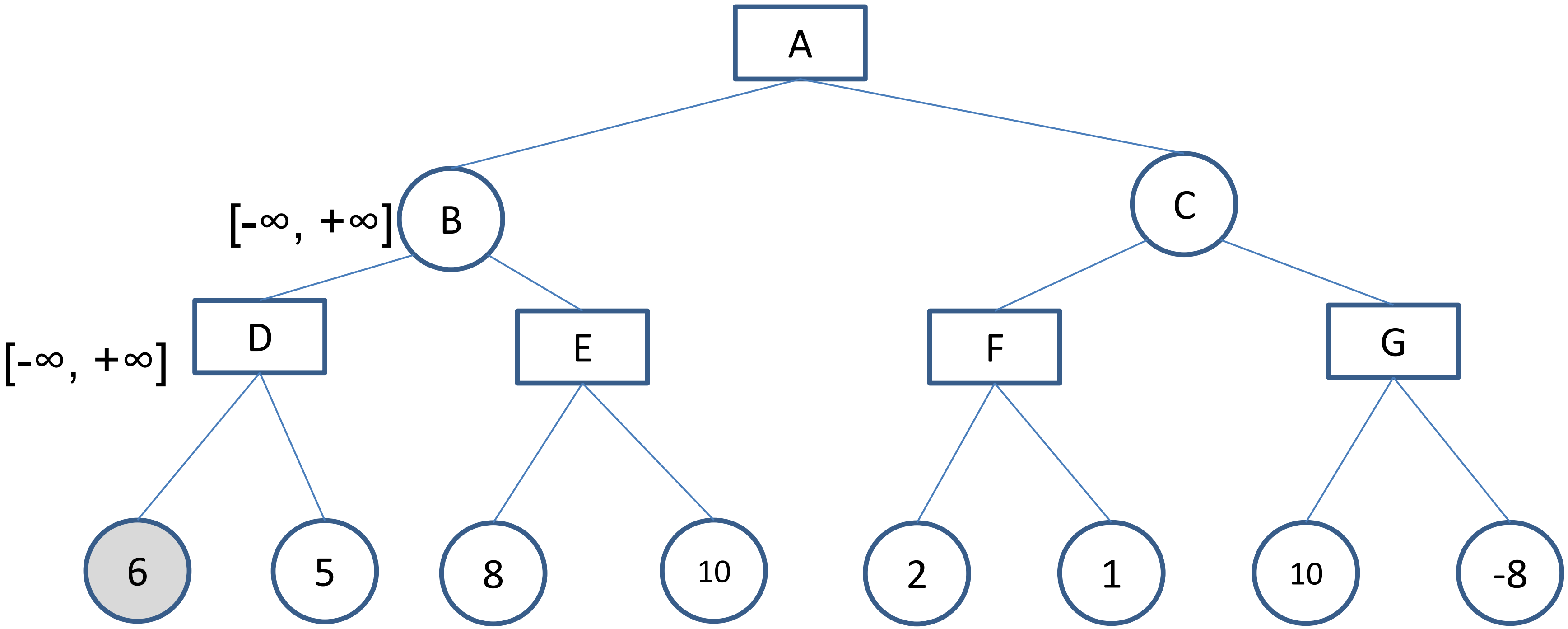
Alpha – Lower bound of Maximizer's value. Perceived value that Maximizer hopes to get with a competitive Minimizer

Beta – Upper Bound of Minimizer's value.



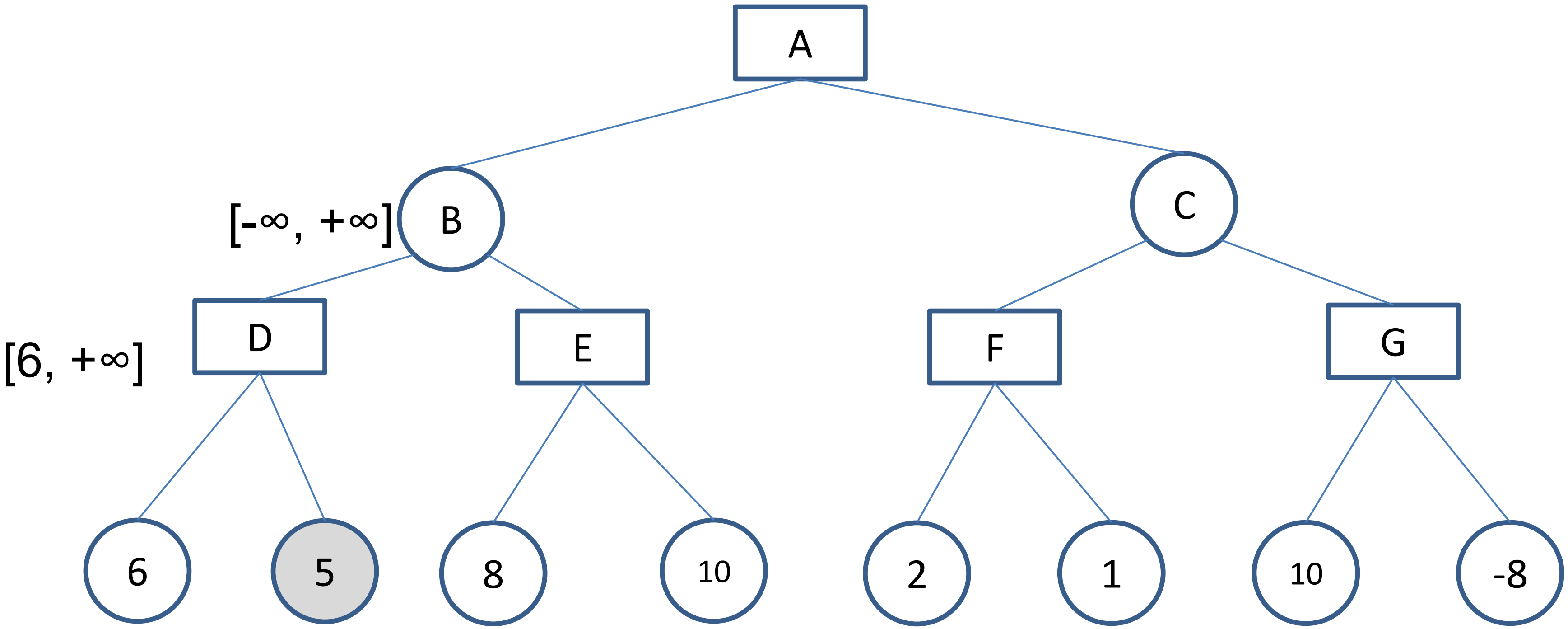
Alpha – beta Pruning – Example -3

$[\ \alpha\ ,\ \beta\ ] = [-\infty,\ +\infty]$



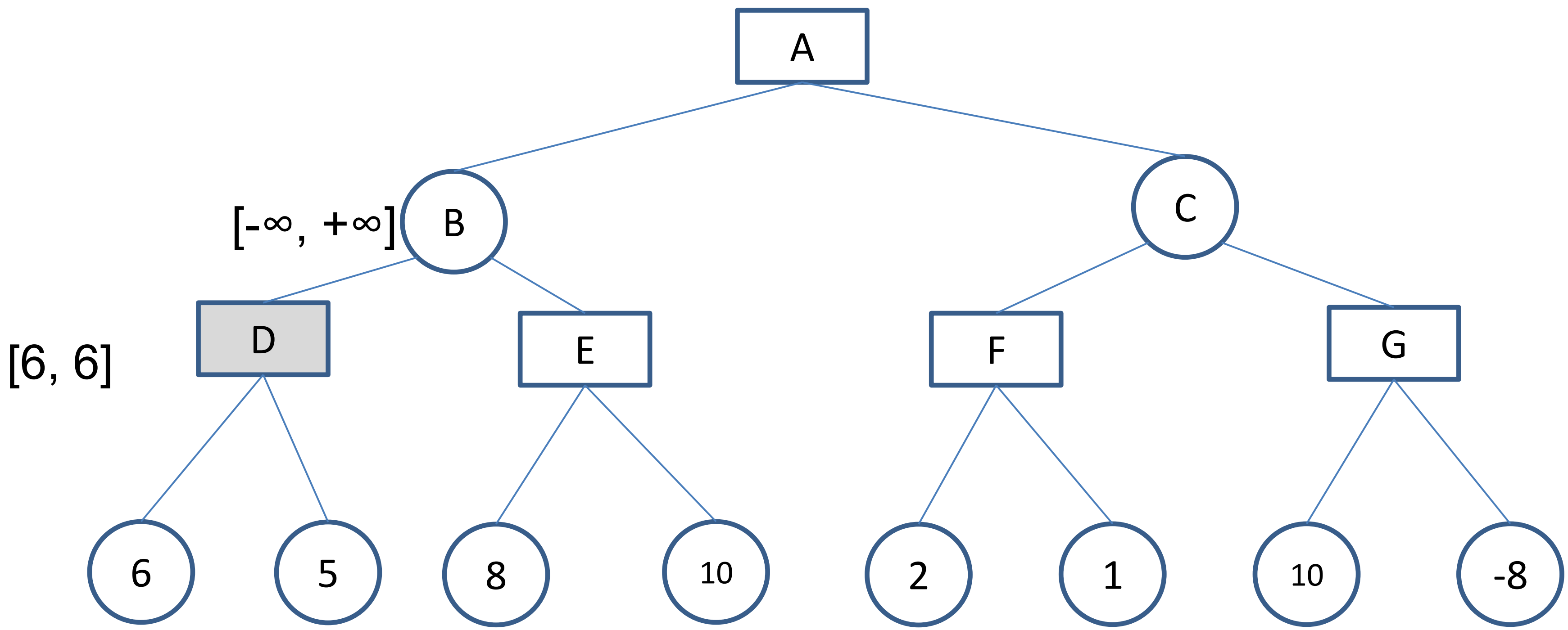
Alpha – beta Pruning – Example -3

$[\alpha, \beta] = [-\infty, +\infty]$



Alpha – beta Pruning – Example -3

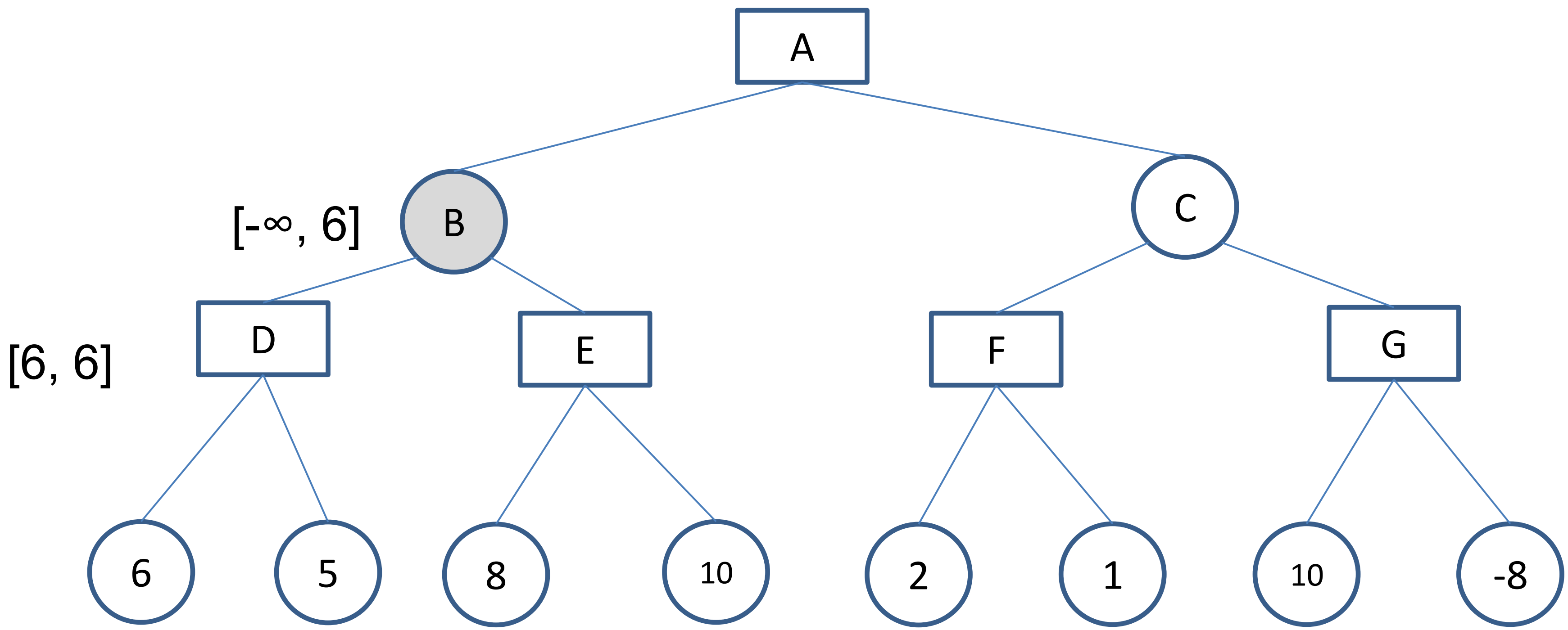
$[\alpha, \beta] = [-\infty, +\infty]$



* As all the successors are analyzed the bounds are same

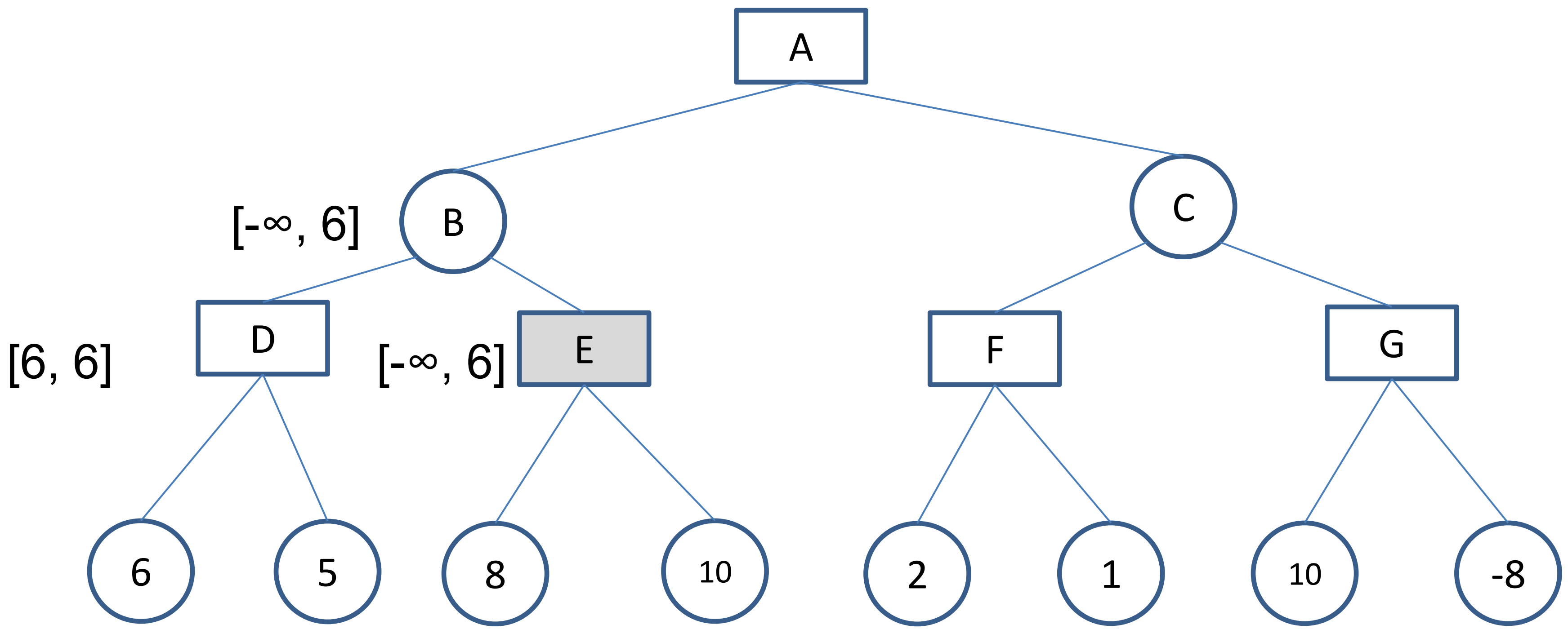
Alpha – beta Pruning – Example -3

$[\alpha, \beta] = [-\infty, +\infty]$



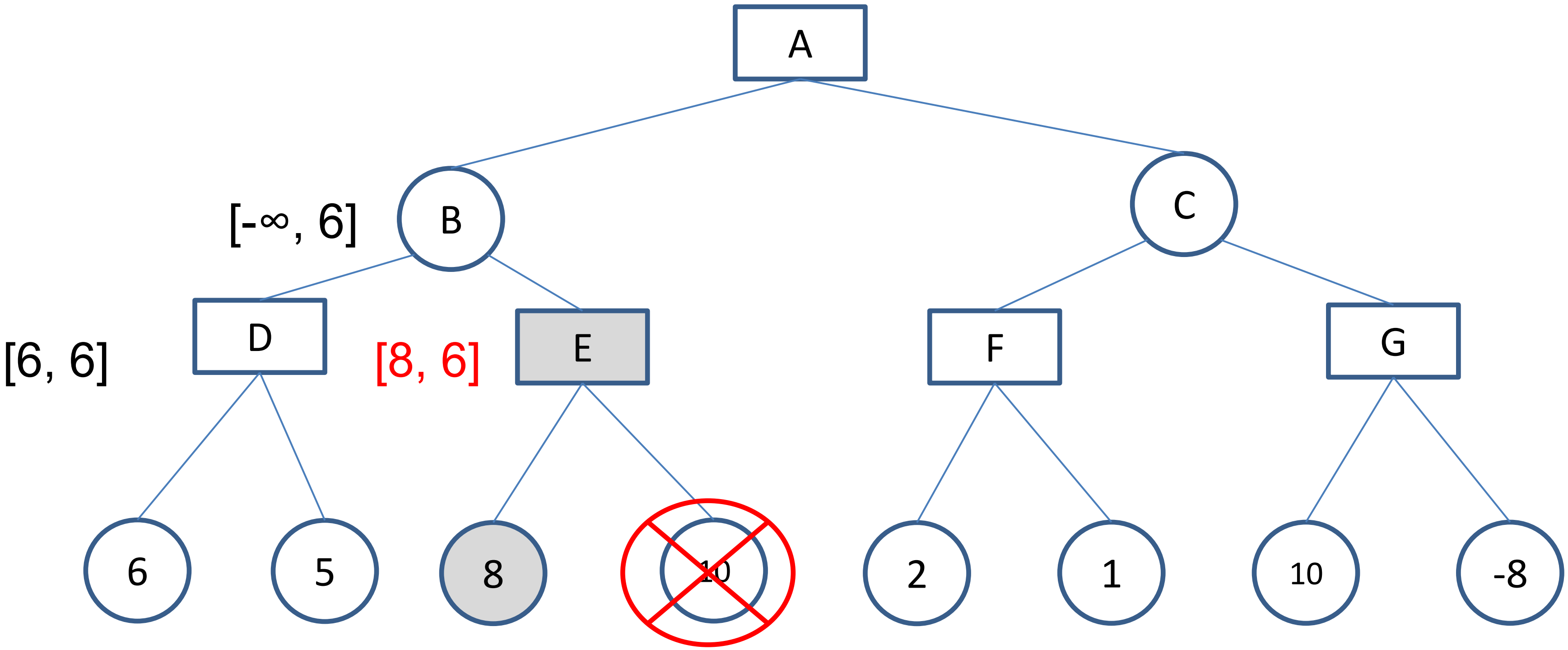
Alpha – beta Pruning – Example -3

$[\,\alpha\,,\,\beta\,] = [-\infty, +\infty]$



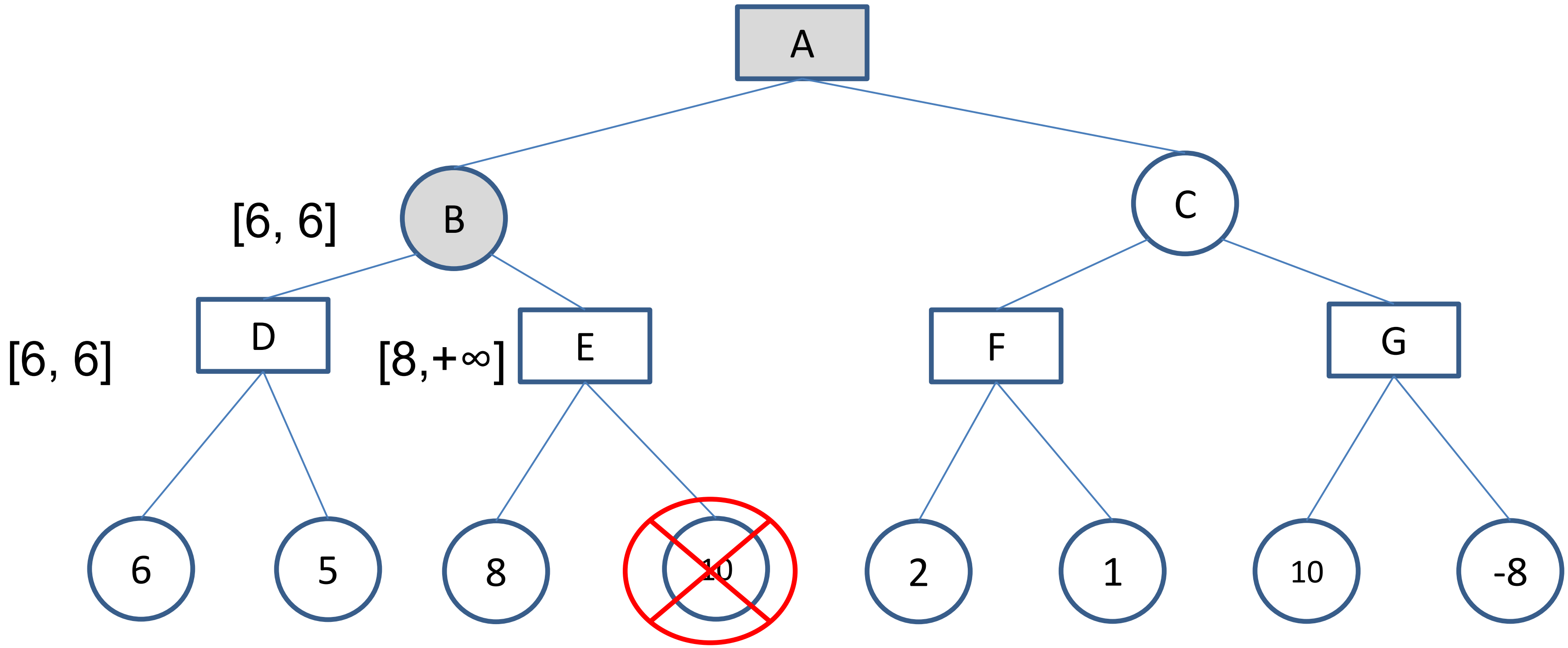
Alpha – beta Pruning – Example -3

$[\ \alpha\ ,\ \beta\ ] = [-\infty, +\infty]$



Alpha – beta Pruning – Example -3

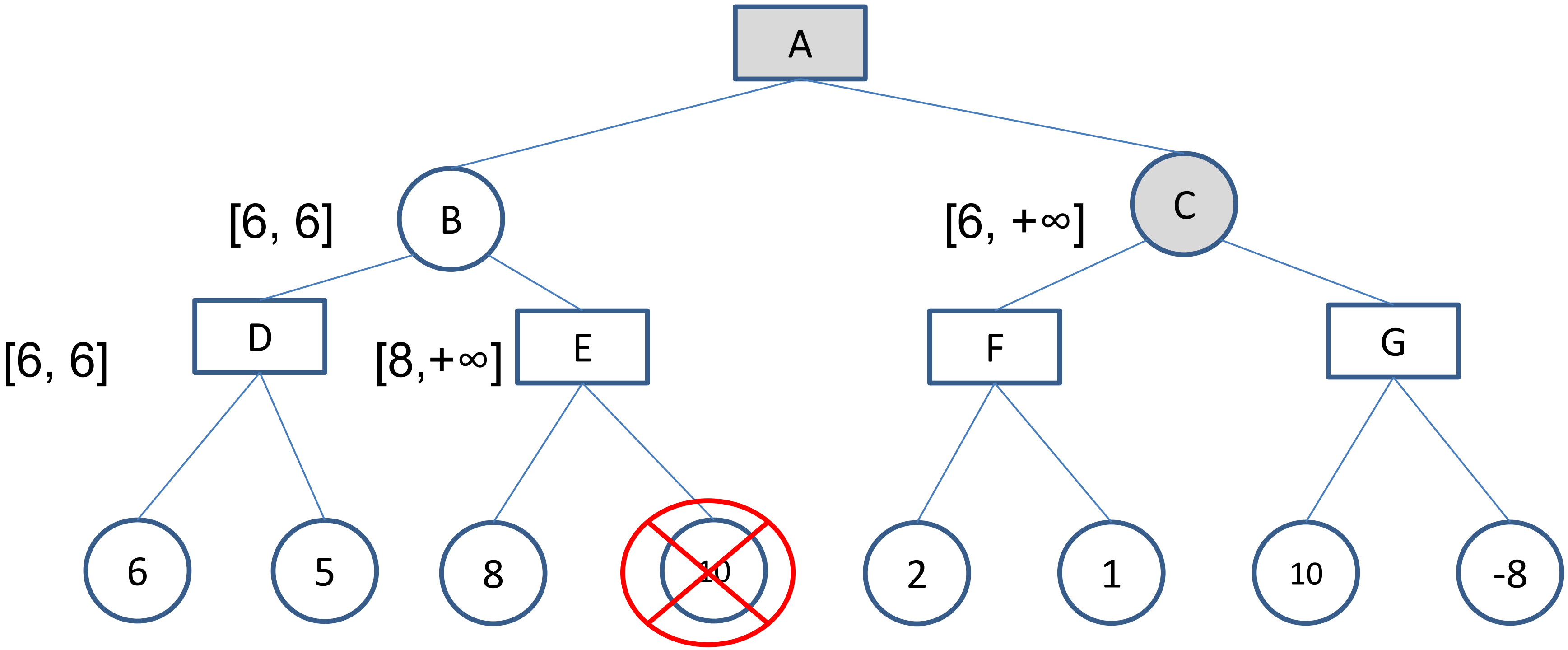
$[\, \alpha \, , \, \beta \,] = \cancel{[-\infty, +\infty]}\ [6, +\infty]$



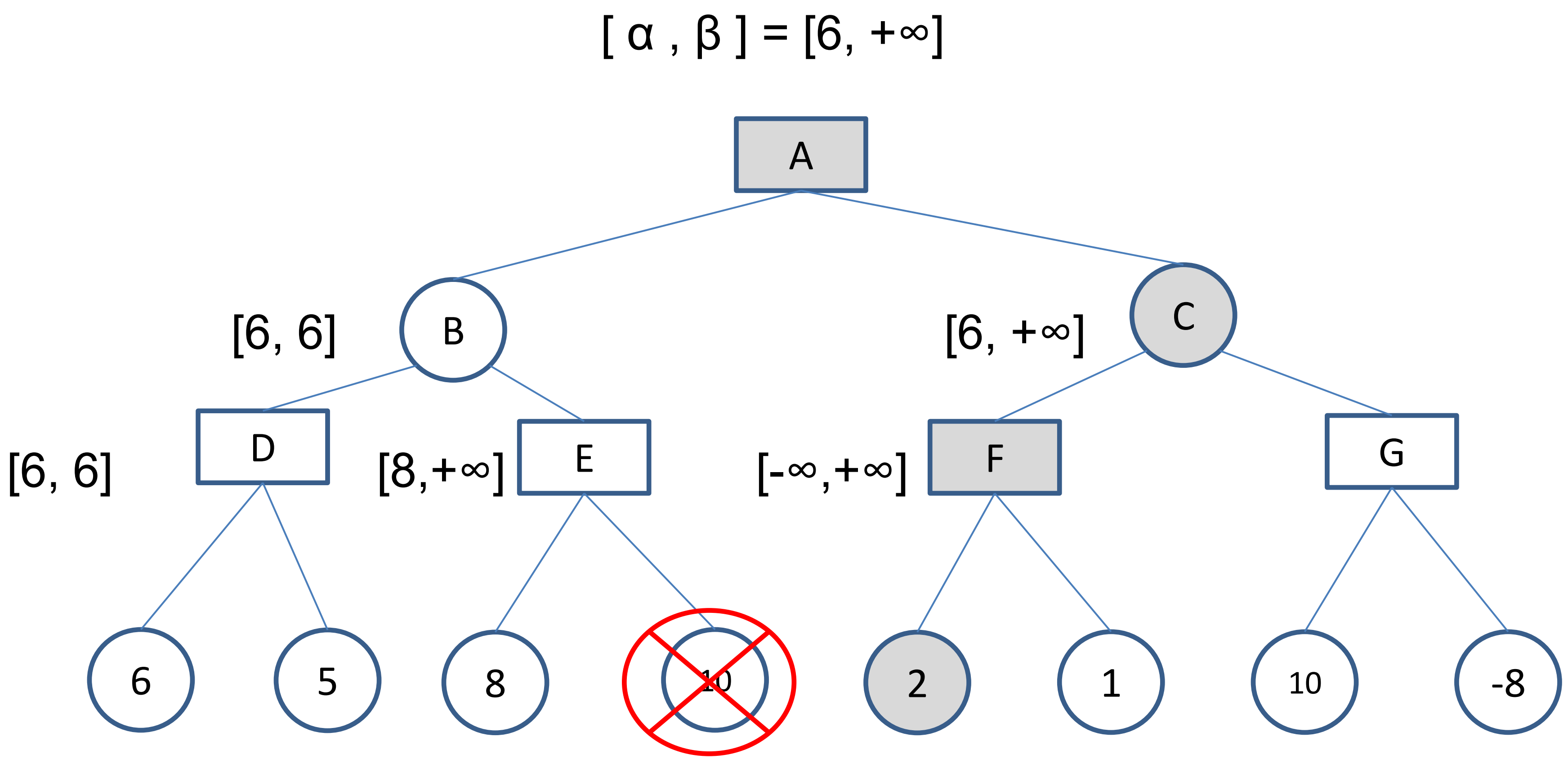
* As successors at MAX are pruned beta is reset

Alpha – beta Pruning – Example -3

$[\ \alpha\ ,\ \beta\ ] = [6, +\infty]$

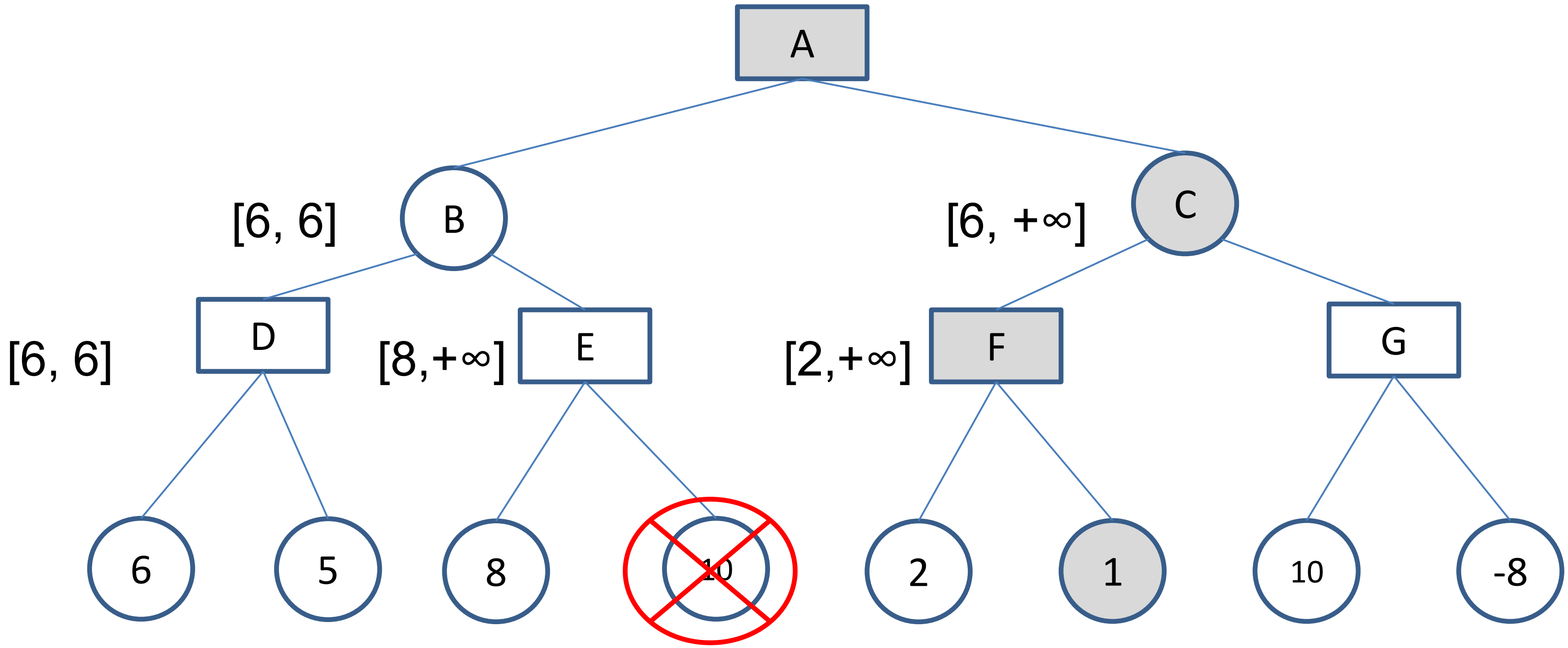


Alpha – beta Pruning – Example -3



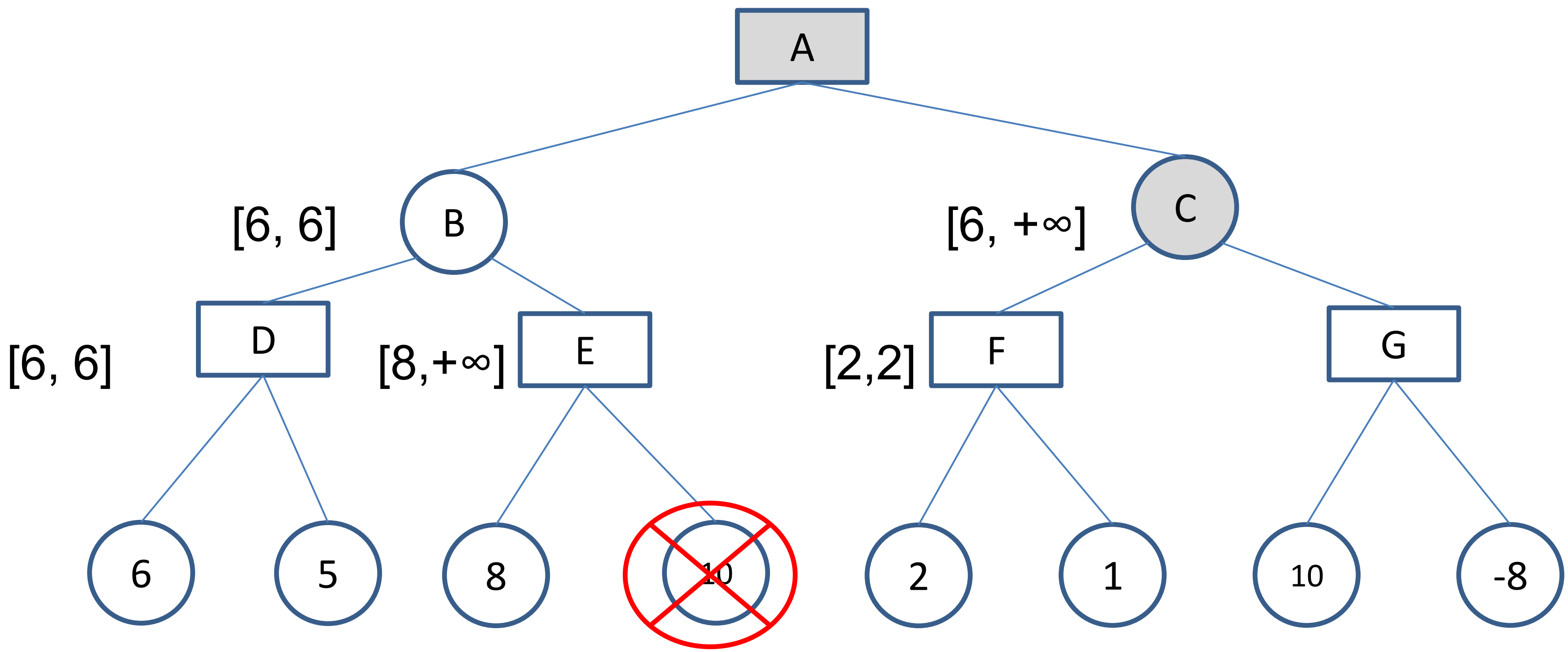
Alpha – beta Pruning – Example -3

$[\alpha, \beta] = [6, +\infty]$

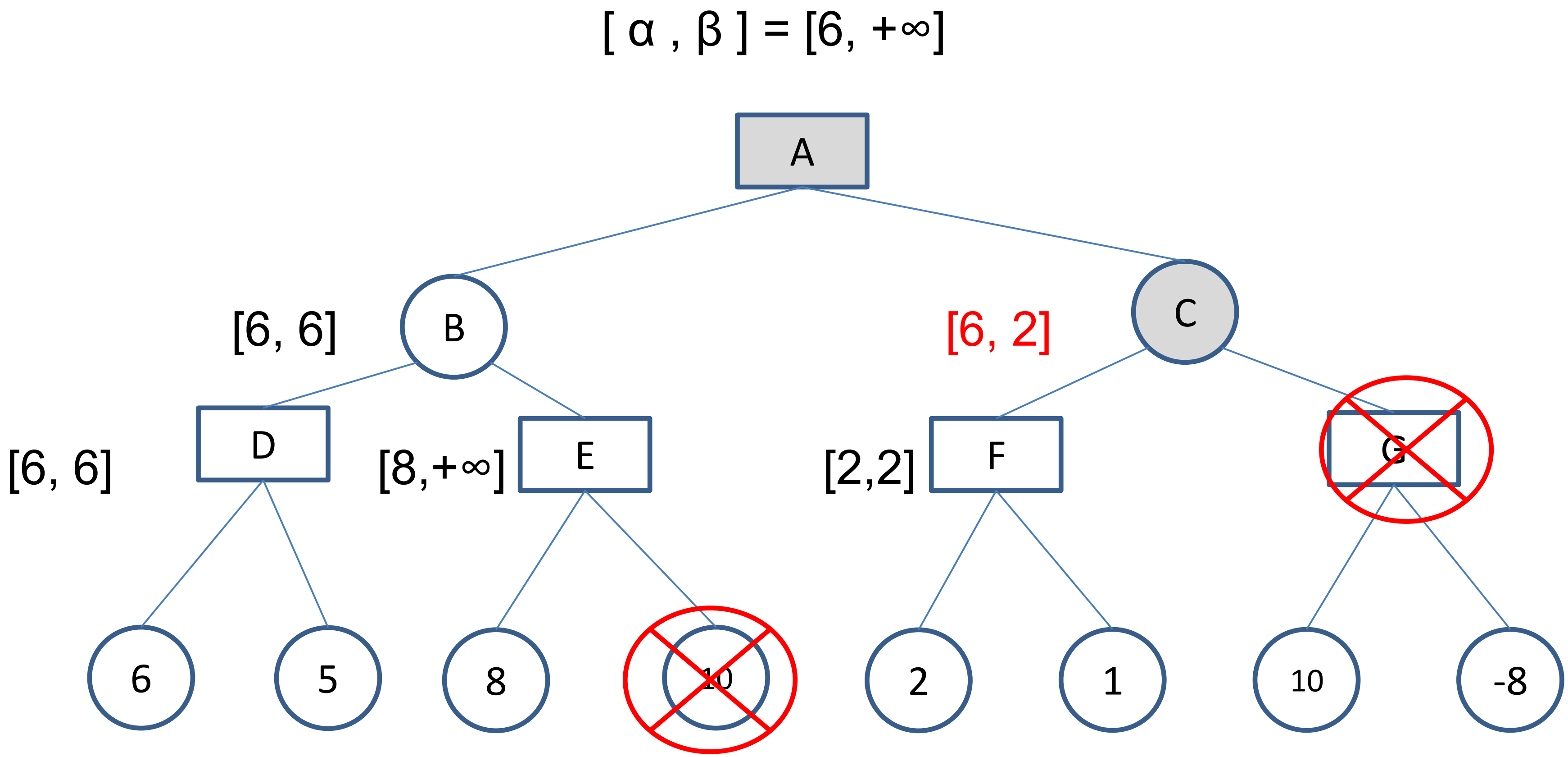


Alpha – beta Pruning – Example -3

$[\alpha, \beta] = [6, +\infty]$

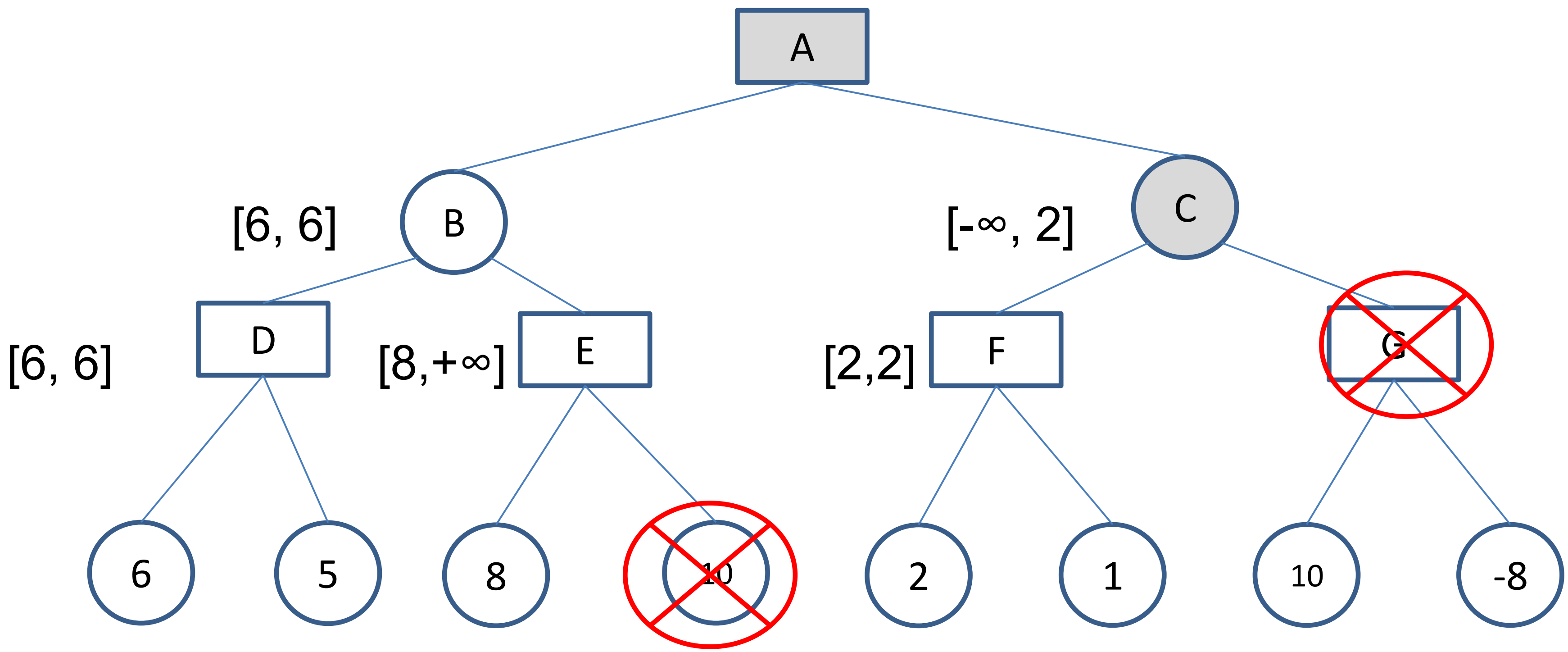


Alpha – beta Pruning – Example -3



Alpha – beta Pruning – Example -3

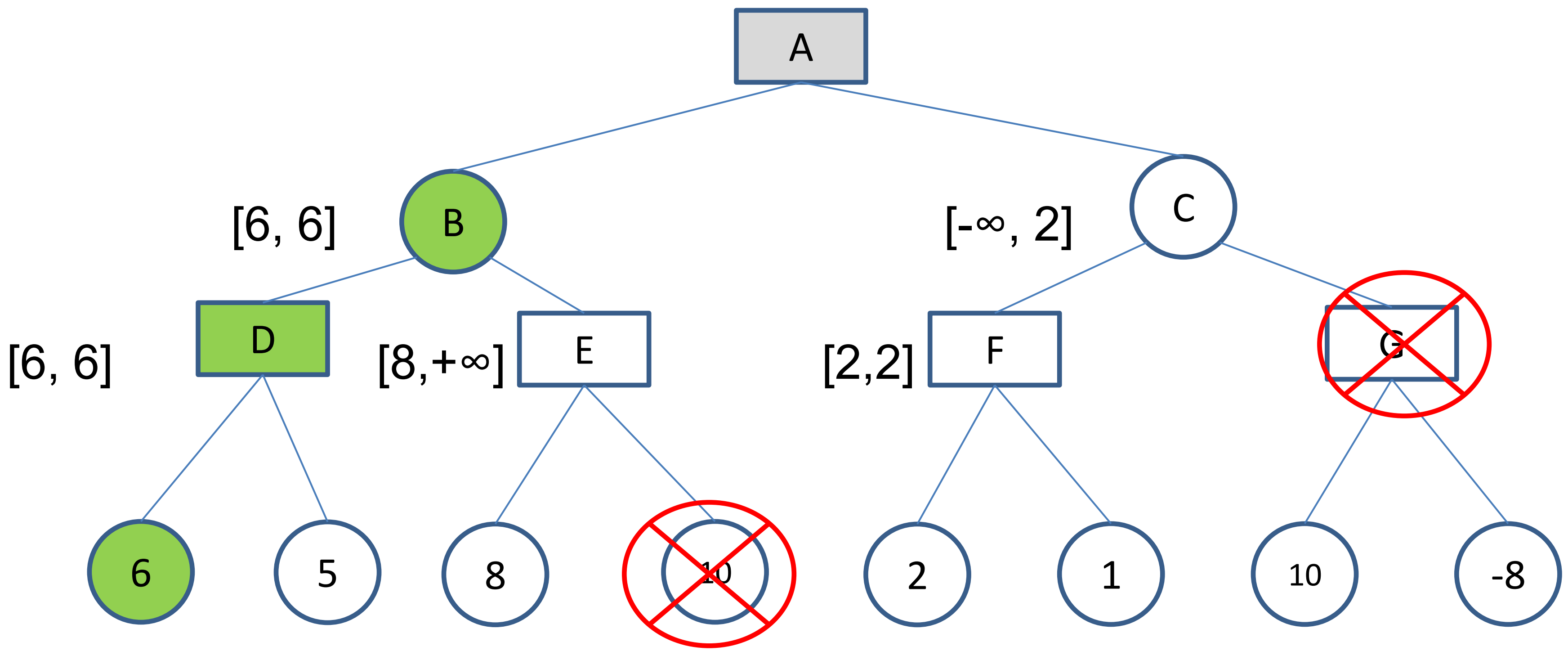
$[\ \alpha\ ,\ \beta\ ] = [6,\ +\infty]$



* As successors at MIN are pruned alpha is reset

Alpha – beta Pruning – Example -3

[α , β] = [6, 6]

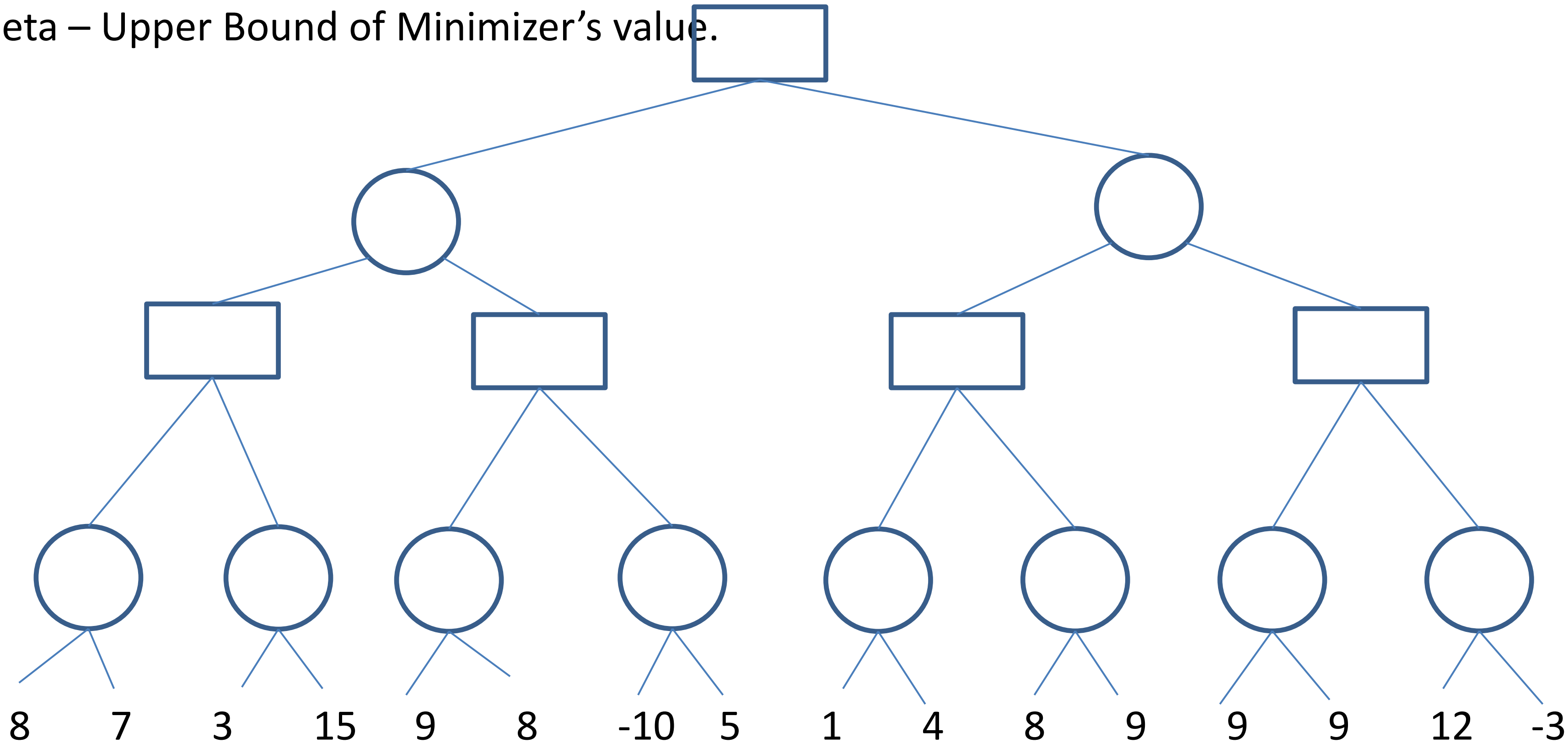


Alpha – beta Pruning – Example -4

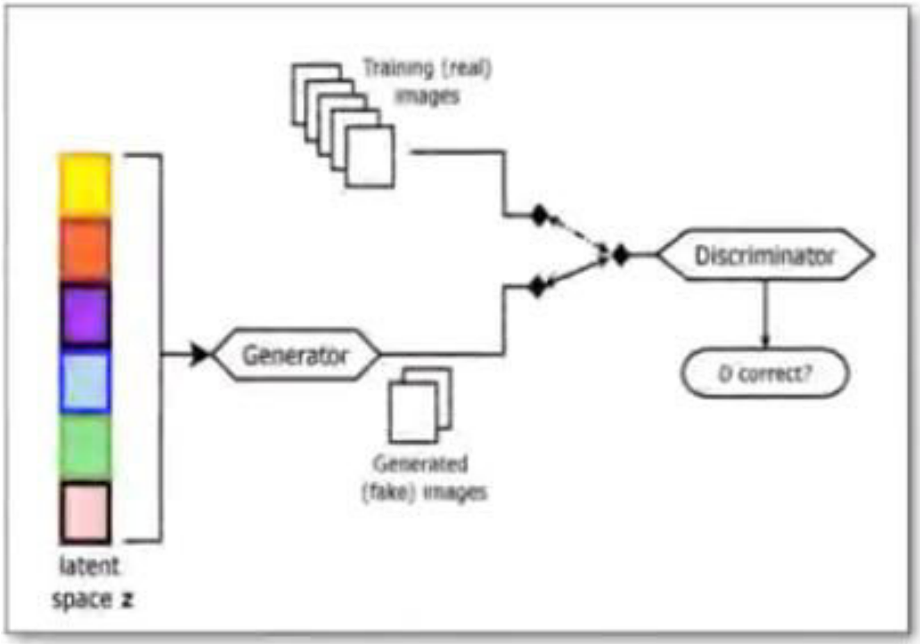
Do for practice.

Alpha – Lower bound of Maximizer's value. Perceived value that Maximizer hopes to get with a competitive Minimizer

Beta – Upper Bound of Minimizer's value.



Games in Image Processing

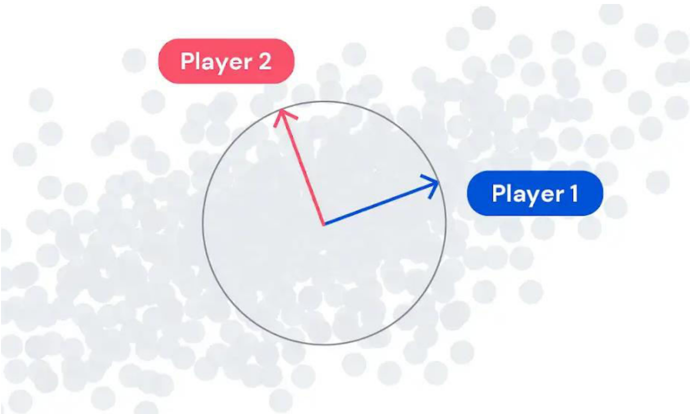


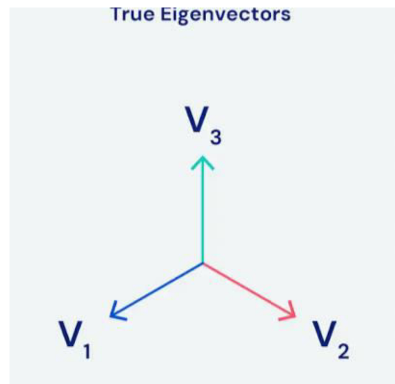
Source Credit:

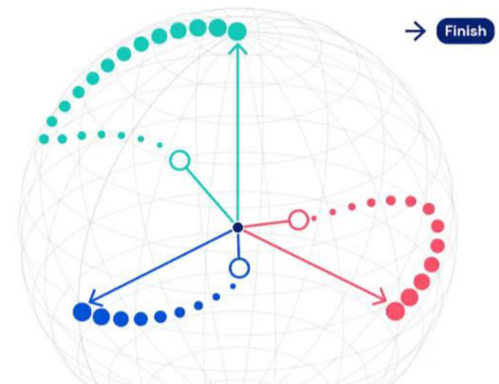
[2019 - Analyzing and Improving the Image Quality of StyleGAN](#)

[Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, Timo Aila](#)

Games in Feature Engineering







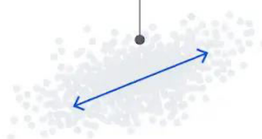
Source Credit:

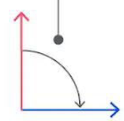
https://deepmind.com/blog/article/EigenGame

2021 - EigenGame: PCA as a Nash Equilibrium , Ian Gemp, Brian McWilliams, Claire Vernade, Thore Graepel

Games in Feature Engineering

$$\mathtt{Utility}(v_i \mid v_{j<i}) = \boxed{\mathtt{Var}(v_i)} - \sum_{j<i} \boxed{\mathtt{Align}}(v_i, v_j)$$





Source Credit:

https://deepmind.com/blog/article/EigenGame

2021 - EigenGame: PCA as a Nash Equilibrium , Ian Gemp, Brian McWilliams, Claire Vernade, Thore Graepel

Required Reading: AIMA - Chapter #5.1, #5.2, #5.3, #5.4

Thank You for all your Attention

Note : Some of the slides are adopted from AIMA TB materials