Chapter 06
Game Playing

Instructor
LE Thanh Sach, Ph.D.

Instructor's Information

LE Thanh Sach, Ph.D.

Office:

Department of Computer Science,
Faculty of Computer Science and Engineering,
HoChiMinh City University of Technology.

Office Address:

268 LyThuongKiet Str., Dist. 10, HoChiMinh City, Vietnam.

E-mail: LTSACH@hcmut.edu.vn

E-home: http://cse.hcmut.edu.vn/~ltsach/

Tel: (+84) 83-864-7256 (Ext: 5839)

Acknowledgment

The slides in this PPT file are composed using the materials supplied by

- **Prof. Stuart Russell and Peter Norvig**: They are currently from University of California, Berkeley. They are also the author of the book "Artificial Intelligence: A Modern Approach", which is used as the textbook for the course

- **Prof. Tom Lenaerts**, from Université Libre de Bruxelles

Outline

- What are games?
- Optimal decisions in games
  - Which strategy leads to success?
- α-β pruning
- Games of imperfect information

Recap

Searching: no adversary		
Solution	**Uninformed**	**Informed**
Path: needed	Breadth-First Search, Depth-First Search, Uninform-Cost Search, Depth-Limited Search, Iterative Deepening Search, etc	Best-First Search, A*, IDA*, RBFS, SMA*, etc --- Heuristic: admissible, consistent
Path: irrelevant		Hill-Climbing, Simulated Annealing, Genetic Algorithm, etc
Assignments satisfies constraints	Backtracking	Backtracking + heuristics --- Heuristic: MRV, Degree, FC, LCV, etc

Today's lecture

Searching: with adversary (Game Play)

- Solution is strategy:
  - Specifying move for every possible opponent reply
  - Examples: chess, checkers, Othello, backgammon
- Time limits force an *approximate* solution
- Evaluation function (add knowledge/heuristic):
  - evaluate "goodness" of game position

Game - Introduction

What are and why study games?

- What is a game?
  - Play for fun
  - Techinically, it is an adversarial search

- Why study games?
  - Fun; historically entertaining
  - Interesting subject of study because they are hard
    - ✓ Chess game:
      average branch factor: 35, each player: 50 moves
      → Search tree: 35^{100} nodes ☹

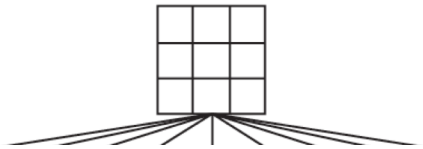
Types of Games

Game setup

- Two players: MAX and MIN
- MAX moves first and they take turns until the game is over. Winner gets award, looser gets penalty.
- Games as search:
  - Initial state: e.g. board configuration of chess
  - Successor function: list of (move,state) pairs specifying legal moves.
  - Terminal test: Is the game finished?
  - Utility function: Gives numerical value of terminal states. E.g. win (+1), loose (-1) and draw (0) in tic-tac-toe (next)
- MAX uses search tree to determine next move.

Example, Tic-Tac-Toe

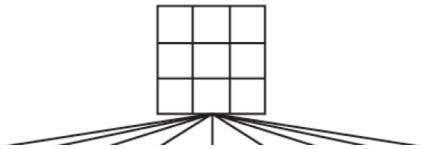
MAX (**x**)



Where should MAX put x on the game board?

Example, Tic-Tac-Toe

MAX (**x**)



Where should MAX put x on the game board?

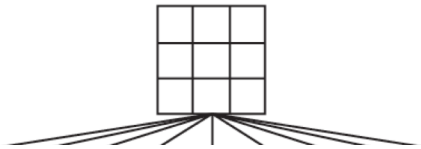
General approach:

- Explore the whole game's tree (if possible) and then make a move

- Look forward several plies, do some evaluation and then make a move (like human)

Example, Tic-Tac-Toe

MAX (**x**)

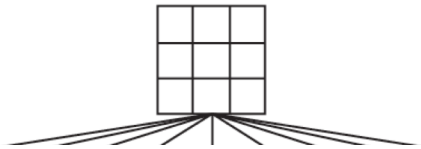


Where should MAX put x on the game board?

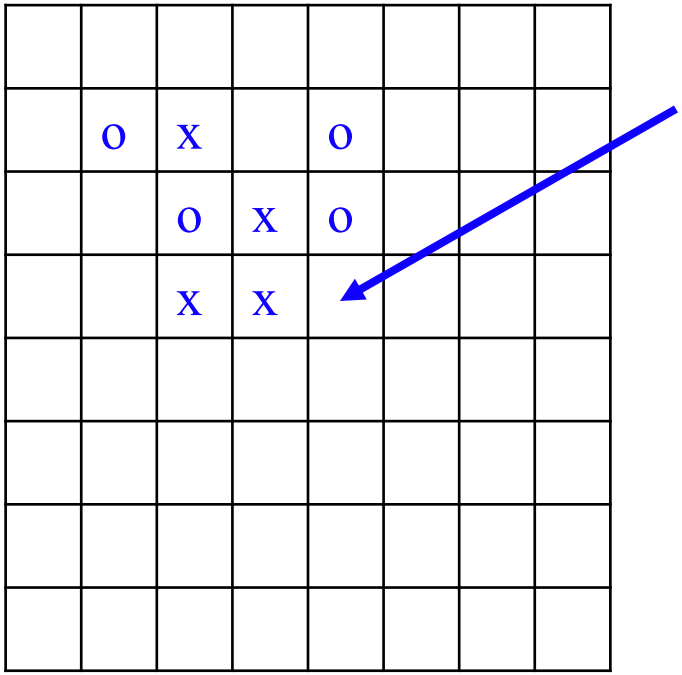
	o	x		o			
		o	x	o			
		x	x				

Example, Tic-Tac-Toe

MAX (**x**)



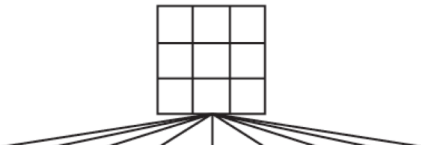
Where should MAX put x on the game board?



x, why?
MAX will have two paths that give it victory

Example, Tic-Tac-Toe

MAX (**x**)



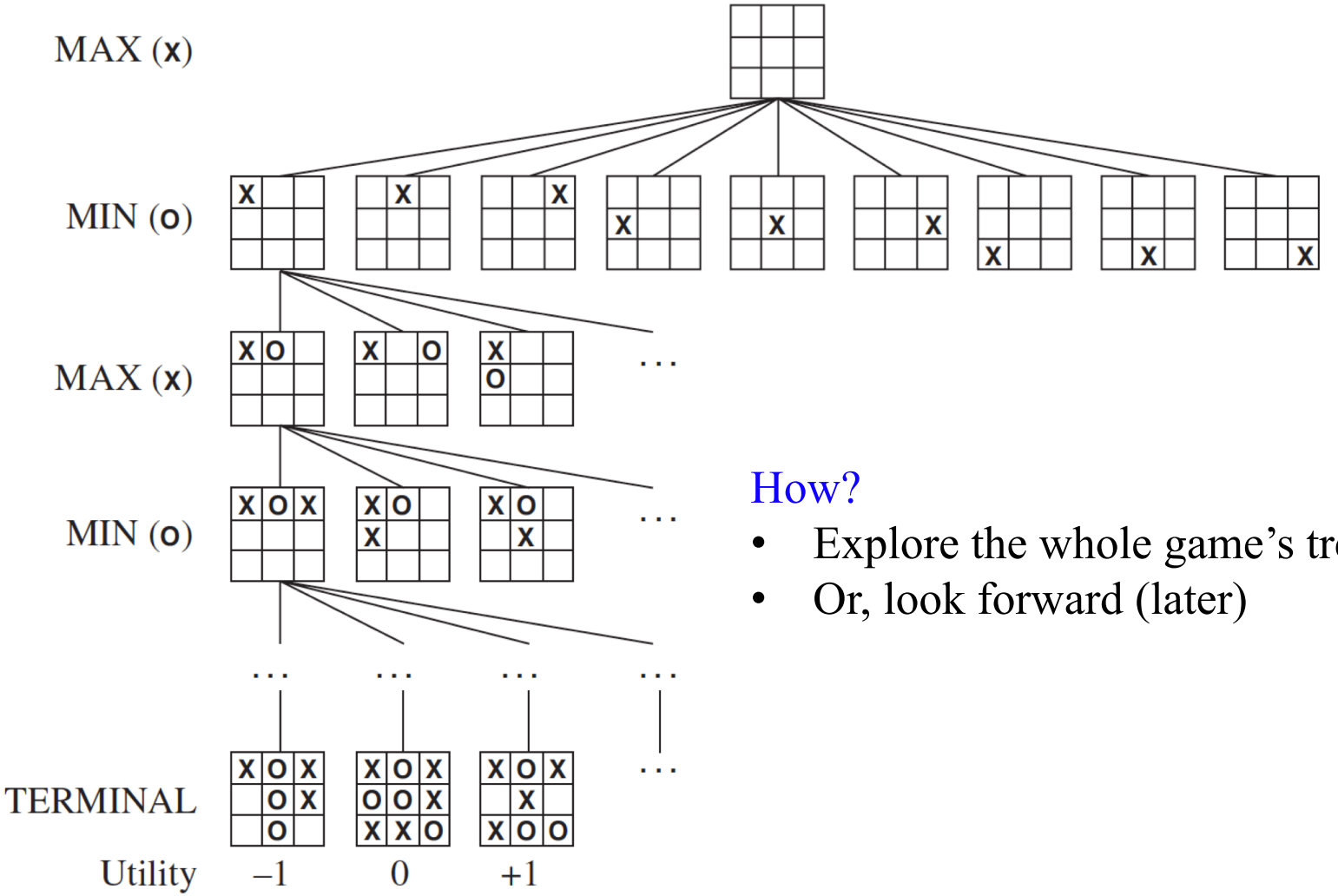
Where should MAX put x on the game board?

	o	x		o			
		o	x	o			
		x	x				

x, why?
MAX will have two paths that give it victory

How?

Example, Tic-Tac-Toe



How?

- Explore the whole game's tree like this
- Or, look forward (later)

Optimal strategies

Optimal strategies

- Find the contingent *strategy* for MAX assuming an infallible MIN opponent.
- Assumption: Both players play optimally !!
- Given a game tree, the optimal strategy can be determined by using the minimax value of each node:

MINIMAX-VALUE(n)=

UTILITY(n)	If n is a terminal
$\max_{s \in successors(n)}$ MINIMAX-VALUE(s)	If n is a max node
$\min_{s \in successors(n)}$ MINIMAX-VALUE(s)	If n is a min node

Optimal strategies

MINIMAX-VALUE(n)=

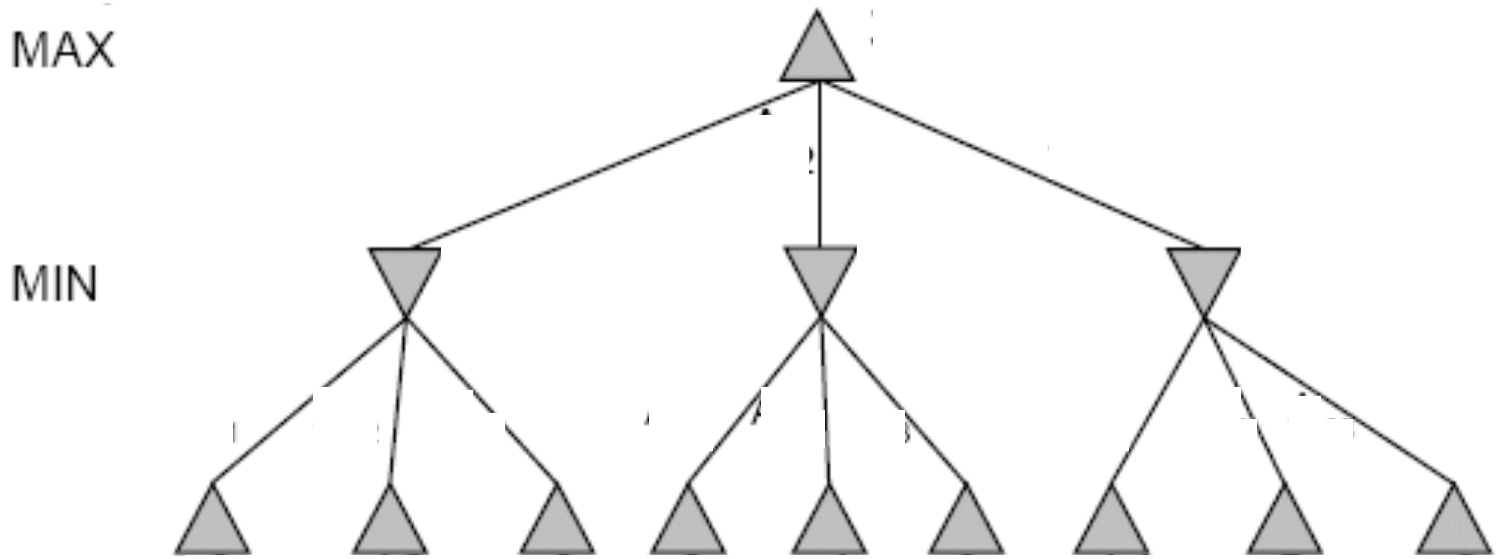
UTILITY(n)	If n is a terminal
$\max_{s \in successors(n)}$ MINIMAX-VALUE(s)	If n is a max node
$\min_{s \in successors(n)}$ MINIMAX-VALUE(s)	If n is a min node

Why?

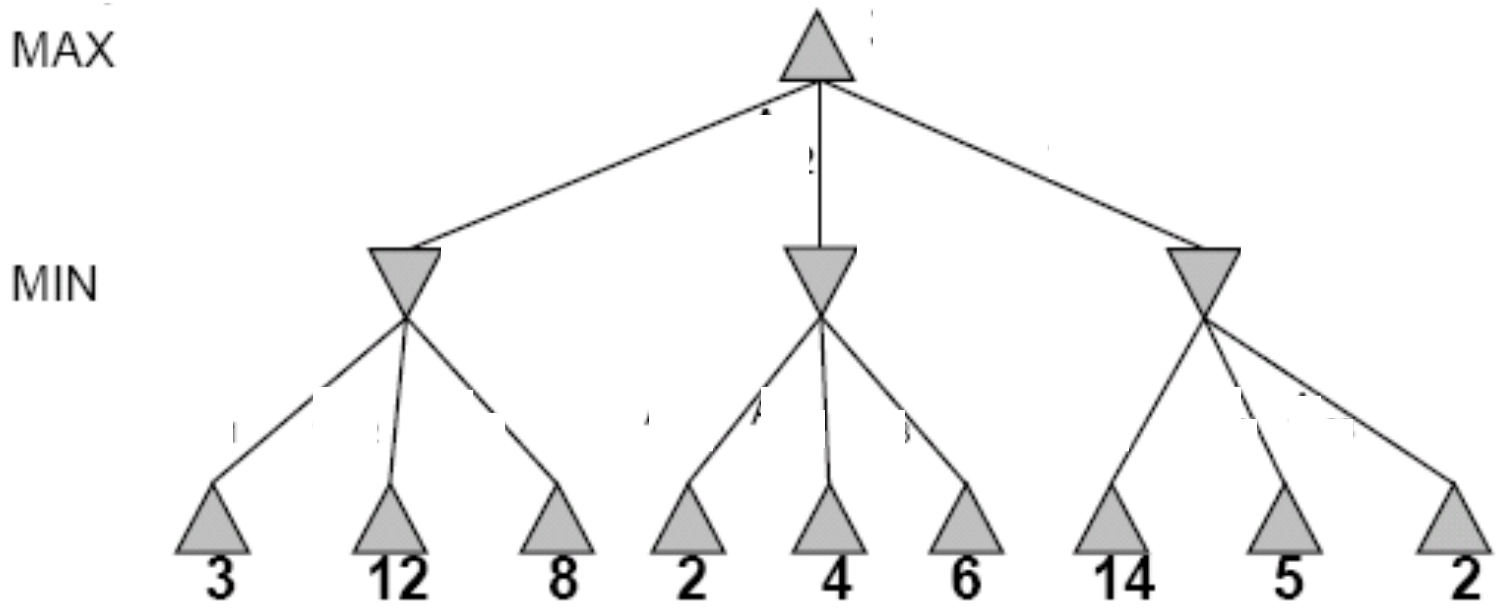
There are two players sharing the same evaluation function f(n)

- With MAX player: the higher f(n) means the better state
- With MIN player: the lower f(n) means the better state
- MAX (MIN) thinks MIN (MAX) is intelligent to play the game optimally

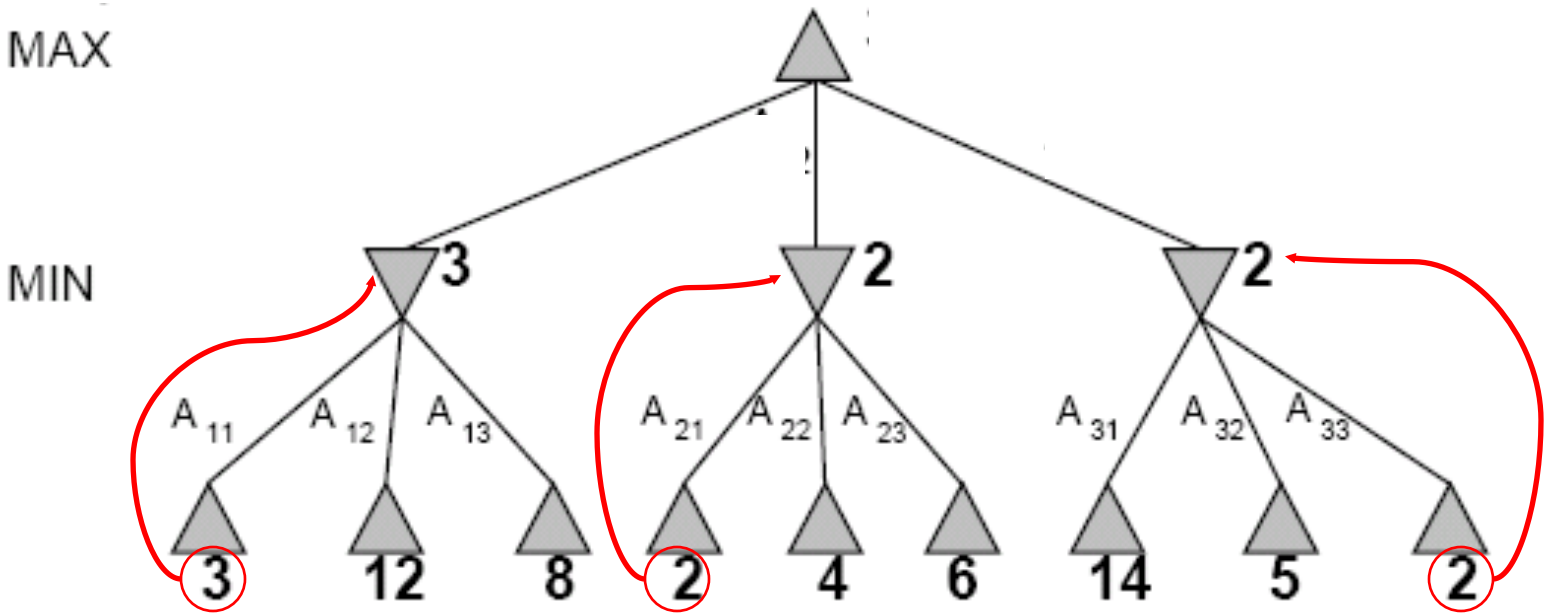
Two-Ply Game Tree



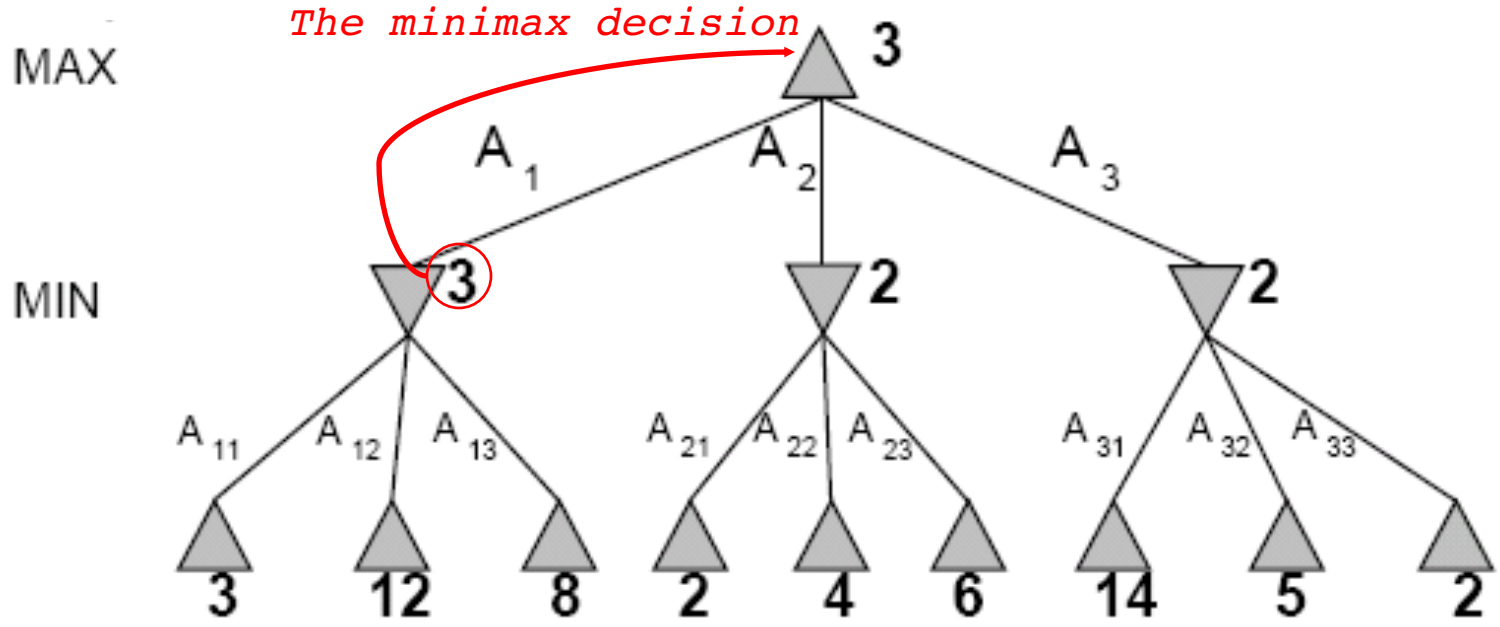
Two-Ply Game Tree



Two-Ply Game Tree



Two-Ply Game Tree

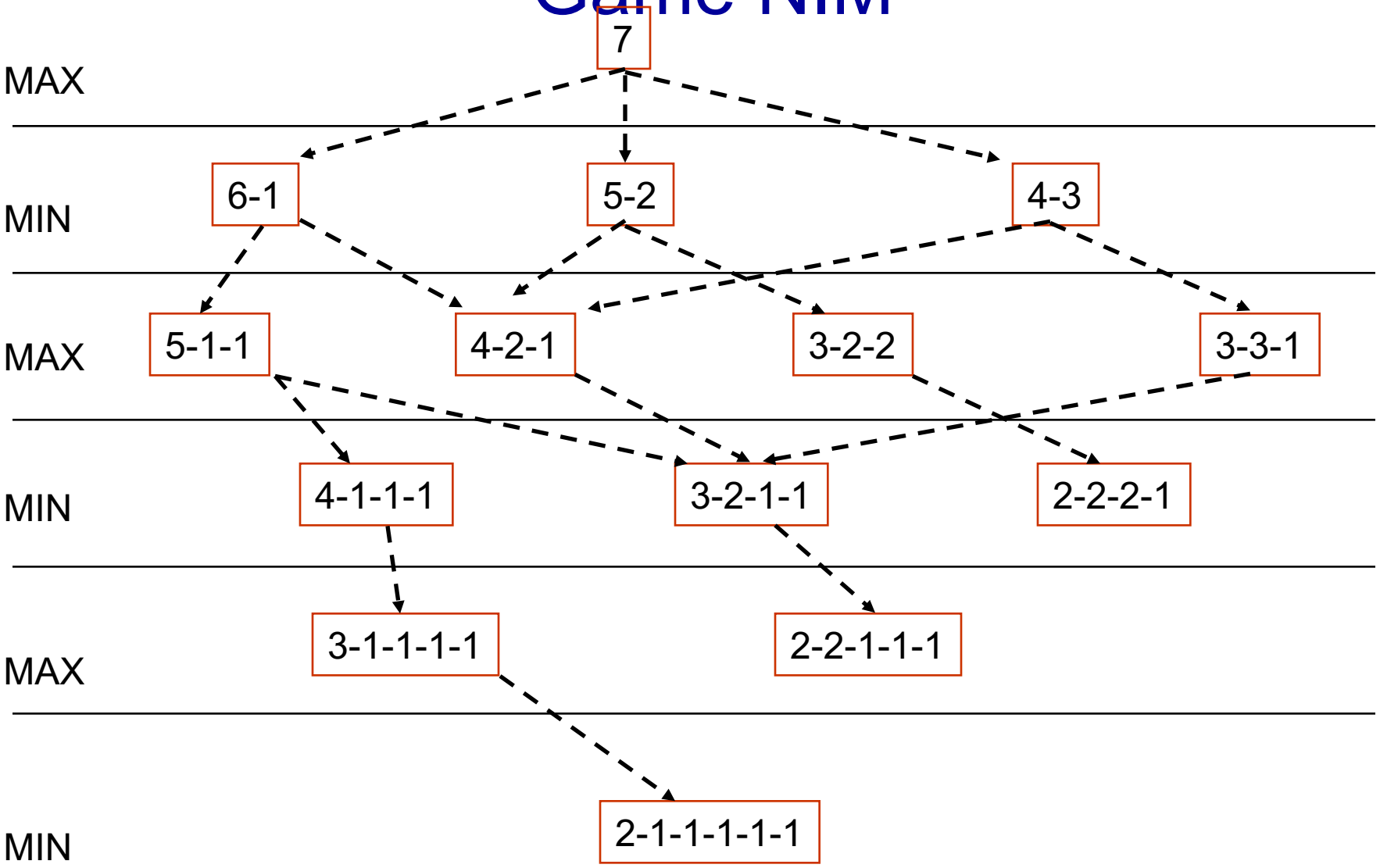


Minimax maximizes the worst-case outcome for max.

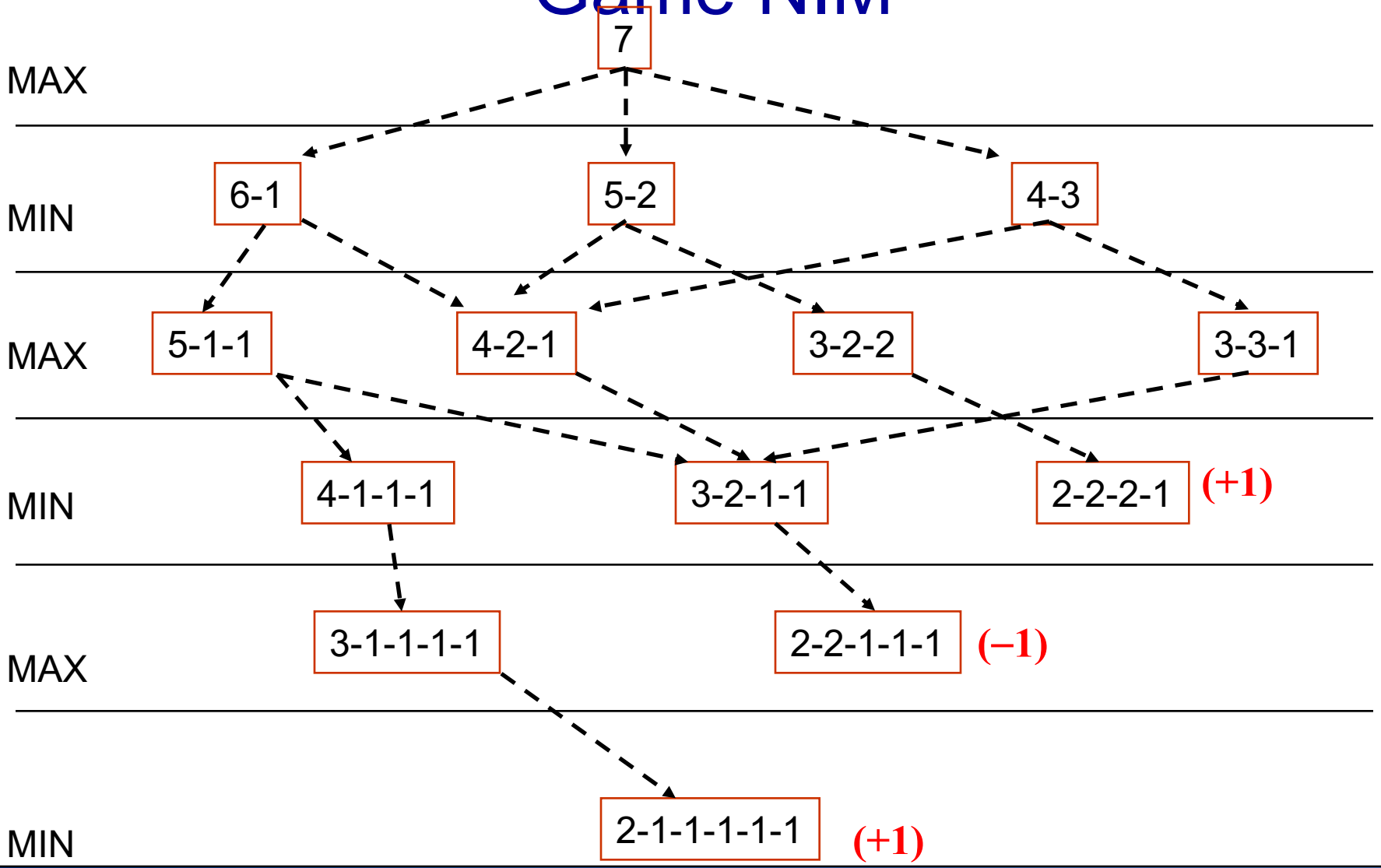
Game NIM

- Given N rods in n (n=1) heaps, two players, take turns
- At each turn, player has to select one heap and separate it into two different heaps having different number of rods
- A player looses the game if he/she can not select a heap for separation.

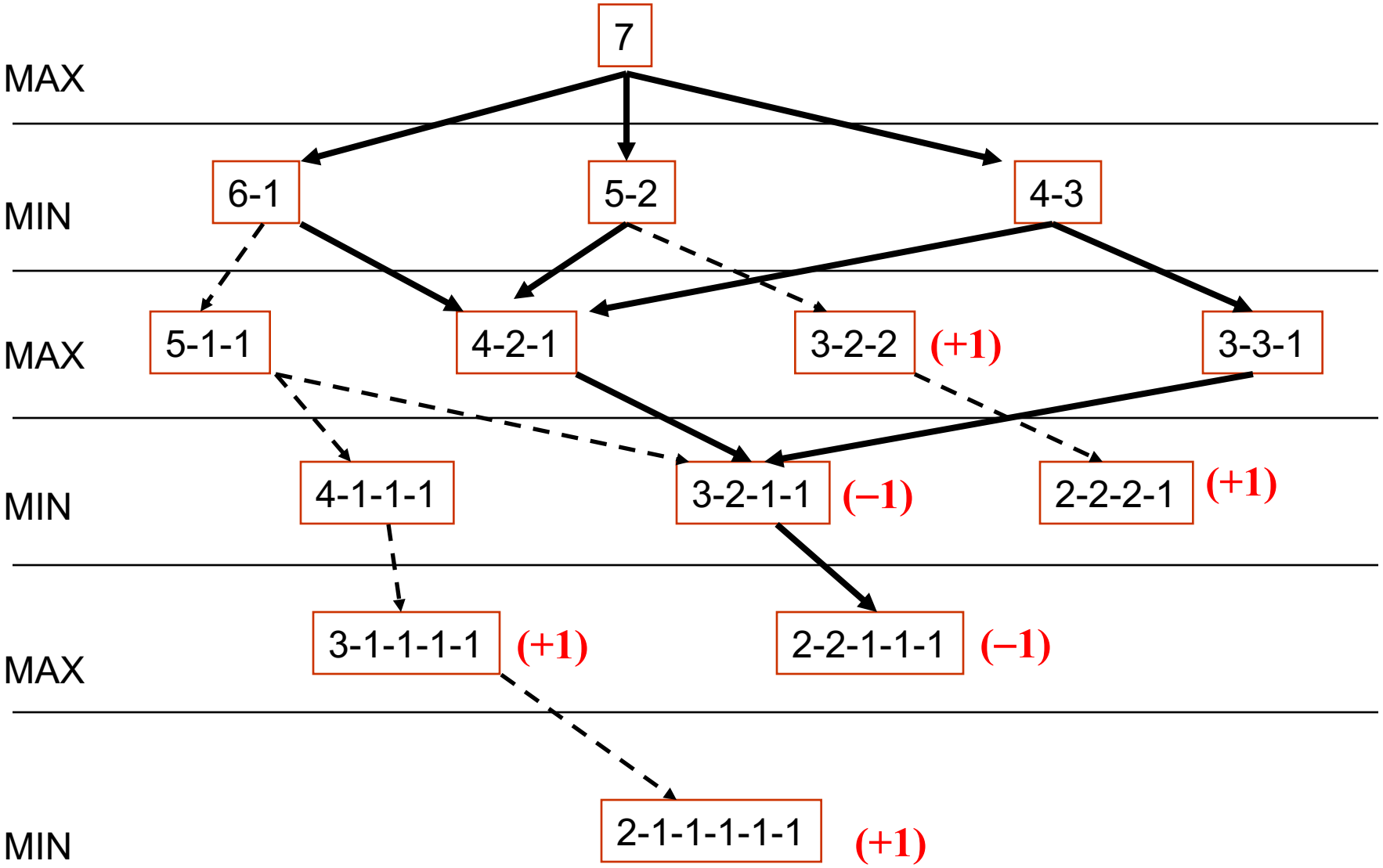
Game NIM


7
MAX
6-1
5-2
4-3
MIN
5-1-1
4-2-1
3-2-2
3-3-1
MAX
4-1-1-1
3-2-1-1
2-2-2-1
MIN
3-1-1-1-1
2-2-1-1-1
MAX
2-1-1-1-1-1
MIN

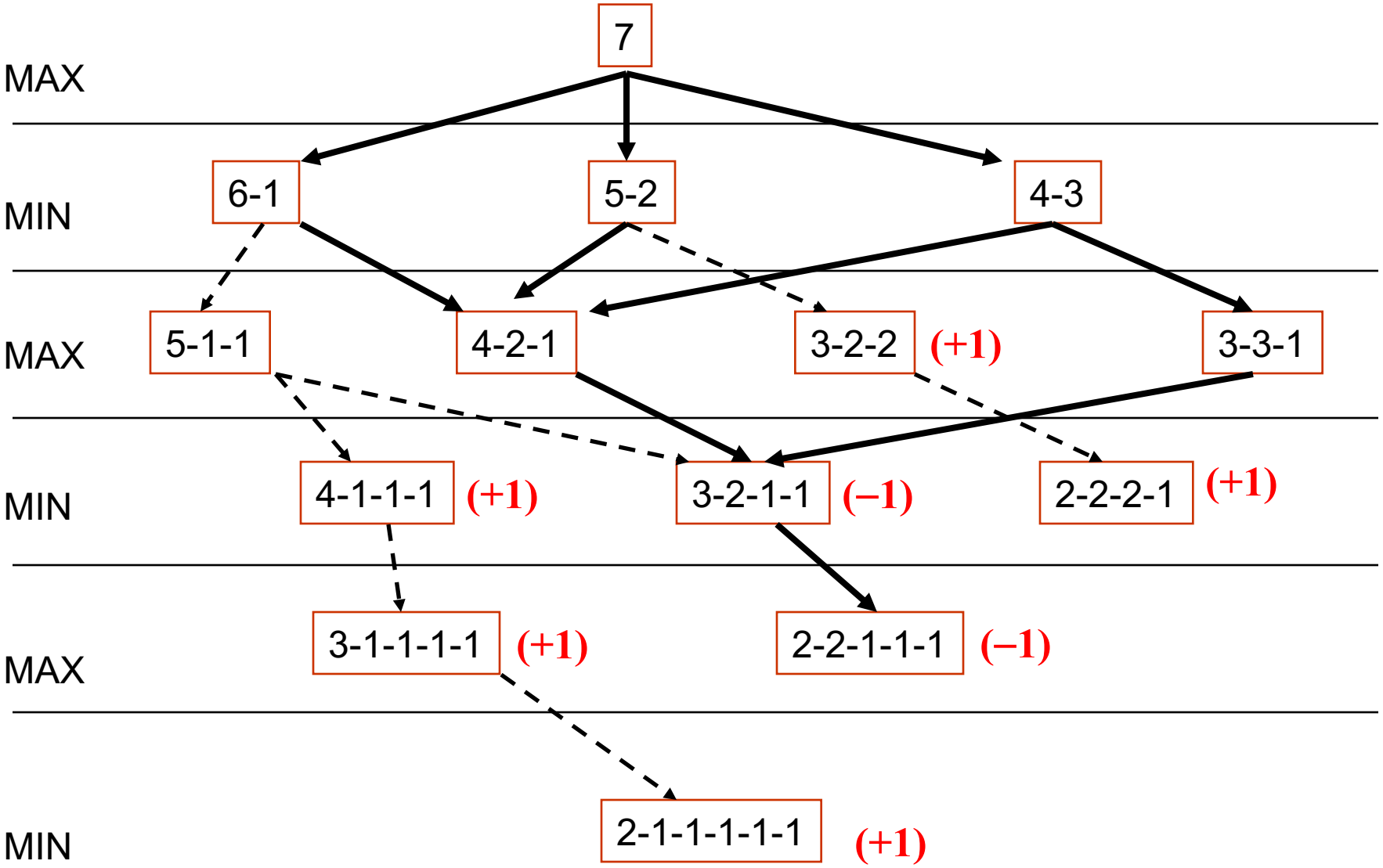
Game NIM


7
MAX
6-1
5-2
4-3
MIN
5-1-1
4-2-1
3-2-2
3-3-1
MAX
4-1-1-1
3-2-1-1
2-2-2-1
(+1)
MIN
3-1-1-1-1
2-2-1-1-1
(–1)
MAX
2-1-1-1-1-1-1
(+1)
MIN

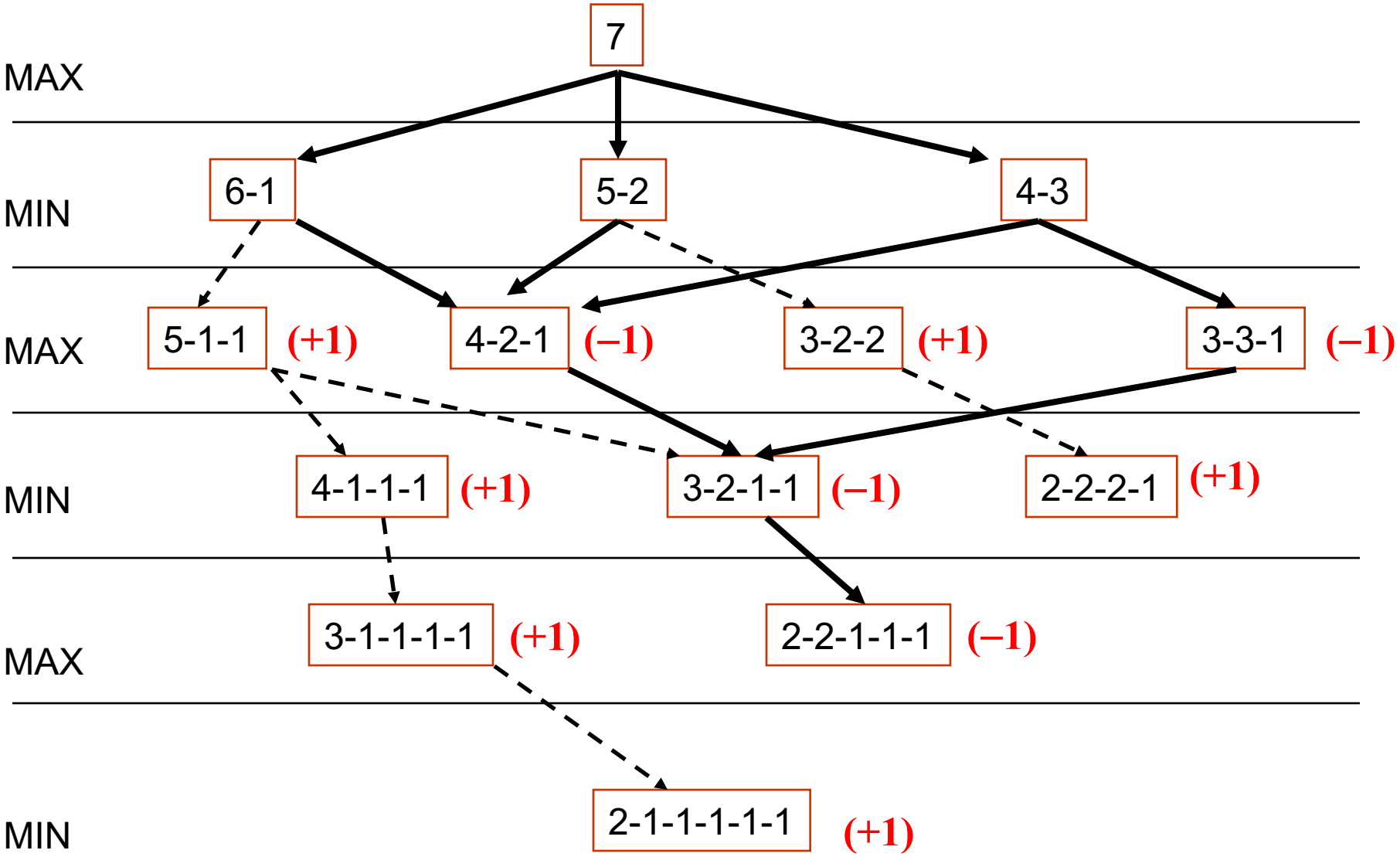
Game NIM


7
MAX
6-1
5-2
4-3
MIN
5-1-1
4-2-1
3-2-2
(+1)
3-3-1
MAX
4-1-1-1
3-2-1-1
(–1)
2-2-2-1
(+1)
MIN
3-1-1-1-1
(+1)
2-2-1-1-1
(–1)
MAX
2-1-1-1-1-1
(+1)
MIN

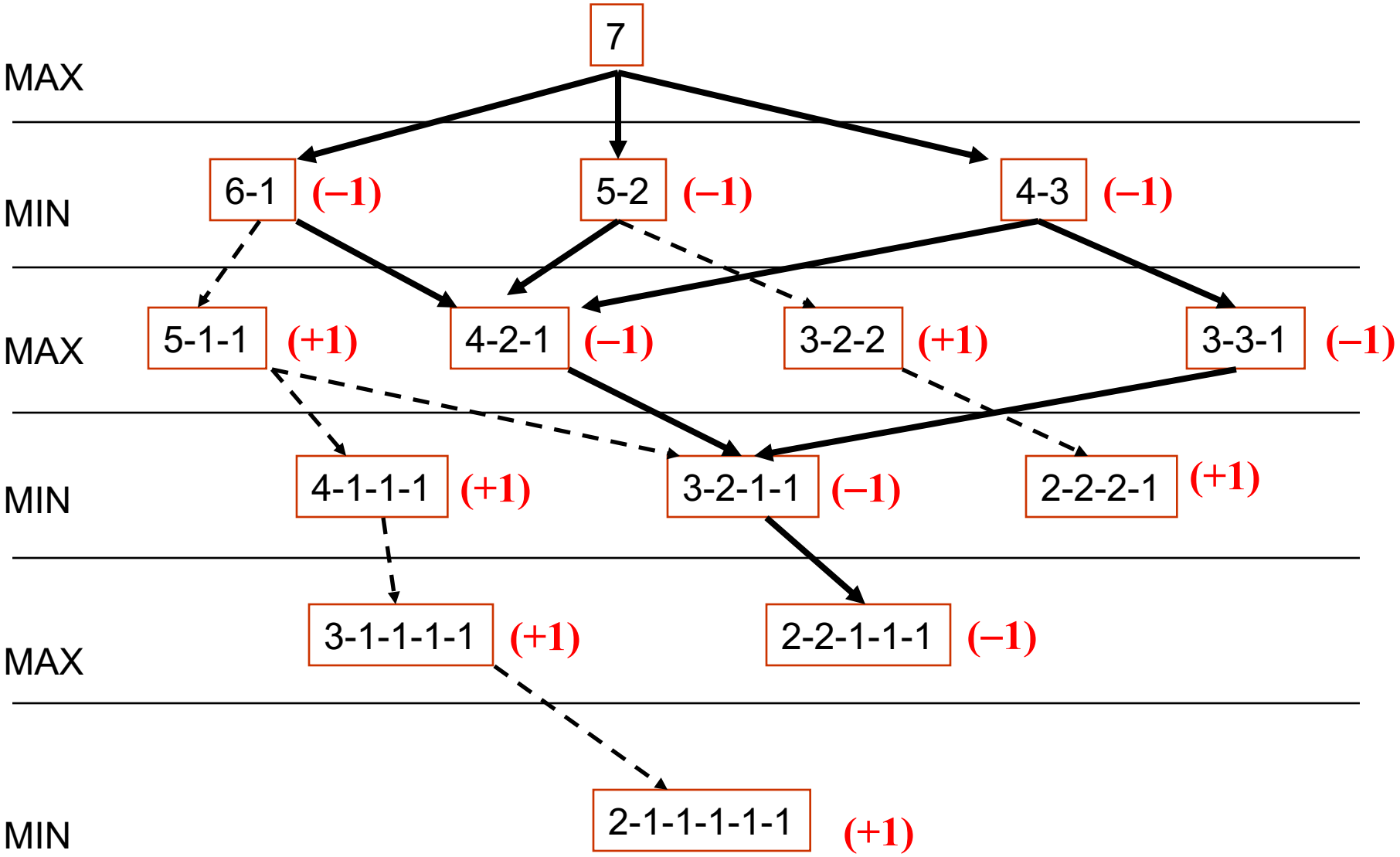
Game NIM


7
MAX
6-1
5-2
4-3
MIN
5-1-1
4-2-1
3-2-2
(+1)
3-3-1
MAX
4-1-1-1
(+1)
3-2-1-1
(–1)
2-2-2-1
(+1)
MIN
3-1-1-1-1
(+1)
2-2-1-1-1
(–1)
MAX
2-1-1-1-1-1
(+1)
MIN

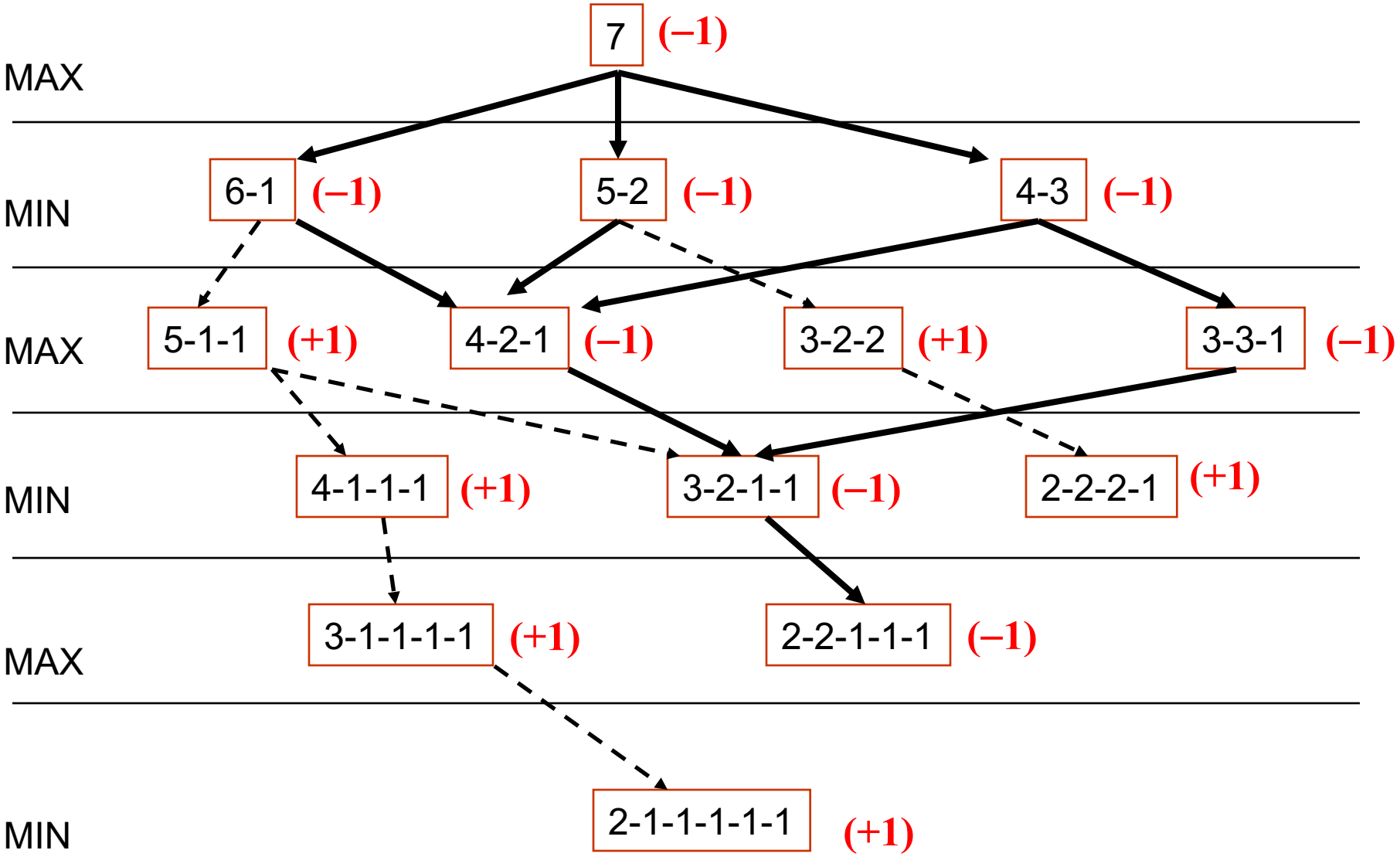
Game NIM


7
MAX
6-1
5-2
4-3
MIN
5-1-1
(+1)
4-2-1
(–1)
3-2-2
(+1)
3-3-1
(–1)
MAX
4-1-1-1
(+1)
3-2-1-1
(–1)
2-2-2-1
(+1)
MIN
3-1-1-1-1
(+1)
2-2-1-1-1
(–1)
MAX
2-1-1-1-1-1
(+1)
MIN

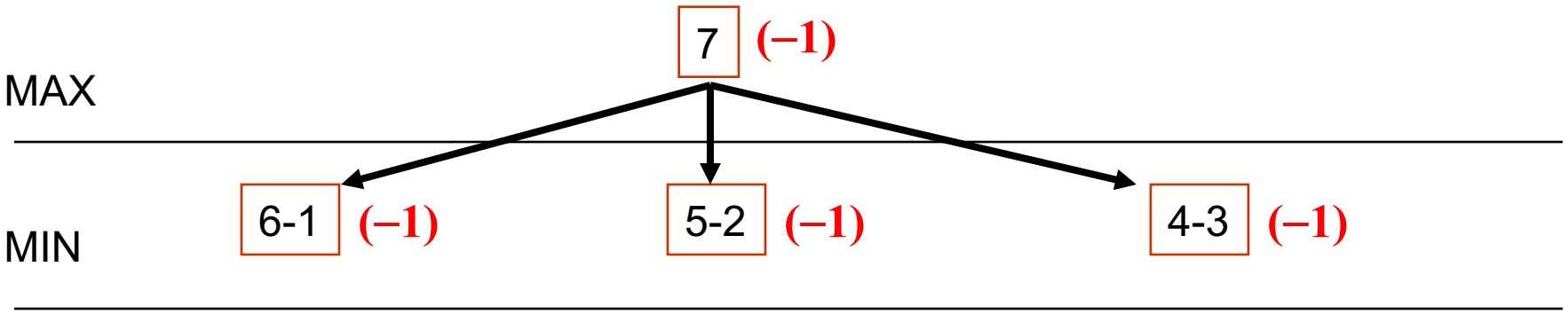
Game NIM


7
MAX
6-1
(–1)
5-2
(–1)
4-3
(–1)
MIN
5-1-1
(+1)
4-2-1
(–1)
3-2-2
(+1)
3-3-1
(–1)
MAX
4-1-1-1
(+1)
3-2-1-1
(–1)
2-2-2-1
(+1)
MIN
3-1-1-1-1
(+1)
2-2-1-1-1
(–1)
MAX
2-1-1-1-1-1
(+1)
MIN

Game NIM


MAX
MIN
MAX
MIN
MAX
MIN
7
(–1)
6-1
(–1)
5-2
(–1)
4-3
(–1)
5-1-1
(+1)
4-2-1
(–1)
3-2-2
(+1)
3-3-1
(–1)
4-1-1-1
(+1)
3-2-1-1
(–1)
2-2-2-1
(+1)
3-1-1-1-1
(+1)
2-2-1-1-1
(–1)
2-1-1-1-1-1
(+1)

Game NIM



MAX: can move to (6,1), (5,2) or (4,3); all have the same chance.

Unluckily for MAX, if MIN plays optimaly, the MAX is impossible to win the this game!

What if MIN does not play optimally?

- Definition of optimal play for MAX assumes MIN plays optimally: maximizes worst-case outcome for MAX.
- But if MIN does not play optimally, MAX will do even better. [Can be proved.]

Minimax Algorithm

function MINIMAX-DECISION(*state*) **returns** *an action*
  inputs: *state*, current state in game
  $v \leftarrow$ 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*)
  $v \leftarrow -\infty$
  for *a,s* in SUCCESSORS(*state*) **do**
    $v \leftarrow$ MAX(*v*,MIN-VALUE(*s*))
  return *v*

---

function MIN-VALUE(*state*) **returns** *a utility value*
  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  $v \leftarrow +\infty$
  for *a,s* in SUCCESSORS(*state*) **do**
    $v \leftarrow$ MIN(*v*,MAX-VALUE(*s*))
  return *v*

Minimax Algorithm

function MINIMAX-DECISION(*state*) **returns** *an action*
  inputs: *state*, current state in game
  $v \leftarrow$ MAX-VALUE(*state*)
  return the *action* in SUCCESSORS(*state*) with value *v*

Indirect recursion

function MAX-VALUE(*state*) **returns** *a utility value*
  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  $v \leftarrow -\infty$
  for *a,s* in SUCCESSORS(*state*) **do**
    $v \leftarrow$ MAX(*v*,MIN-VALUE(*s*))
  return *v*

Indirect recursion

function MIN-VALUE(*state*) **returns** *a utility value*
  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  $v \leftarrow +\infty$
  for *a,s* in SUCCESSORS(*state*) **do**
    $v \leftarrow$ MIN(*v*,MAX-VALUE(*s*))
  return *v*

Minimax Algorithm

function MINIMAX-DECISION(*state*) **returns** *an action*
  inputs: *state*, current state in game
  $v \leftarrow$ 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*)
  $v \leftarrow -\infty$
  for *a,s* in SUCCESSORS(*state*) **do**
    $v \leftarrow$ MAX(v,MIN-VALUE(*s*))
  return *v*

Indirect recursion

for each action a, s = state + a

---

function MIN-VALUE(*state*) **returns** *a utility value*
  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  $v \leftarrow +\infty$
  for *a,s* in SUCCESSORS(*state*) **do**
    $v \leftarrow$ MIN(v,MAX-VALUE(*s*))
  return *v*

Indirect recursion

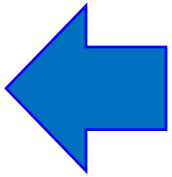
for each action a, s = state + a

Properties of Minimax

Criterion	Minimax
Complete?	Yes ☺
Time	$O(b^m)$ ☹
Space	O(bm) ☺
Optimal?	Yes ☺

Properties of Minimax

Criterion	Minimax
Complete?	Yes
Time	$O(b^m)$
Space	O(bm)
Optimal?	Yes

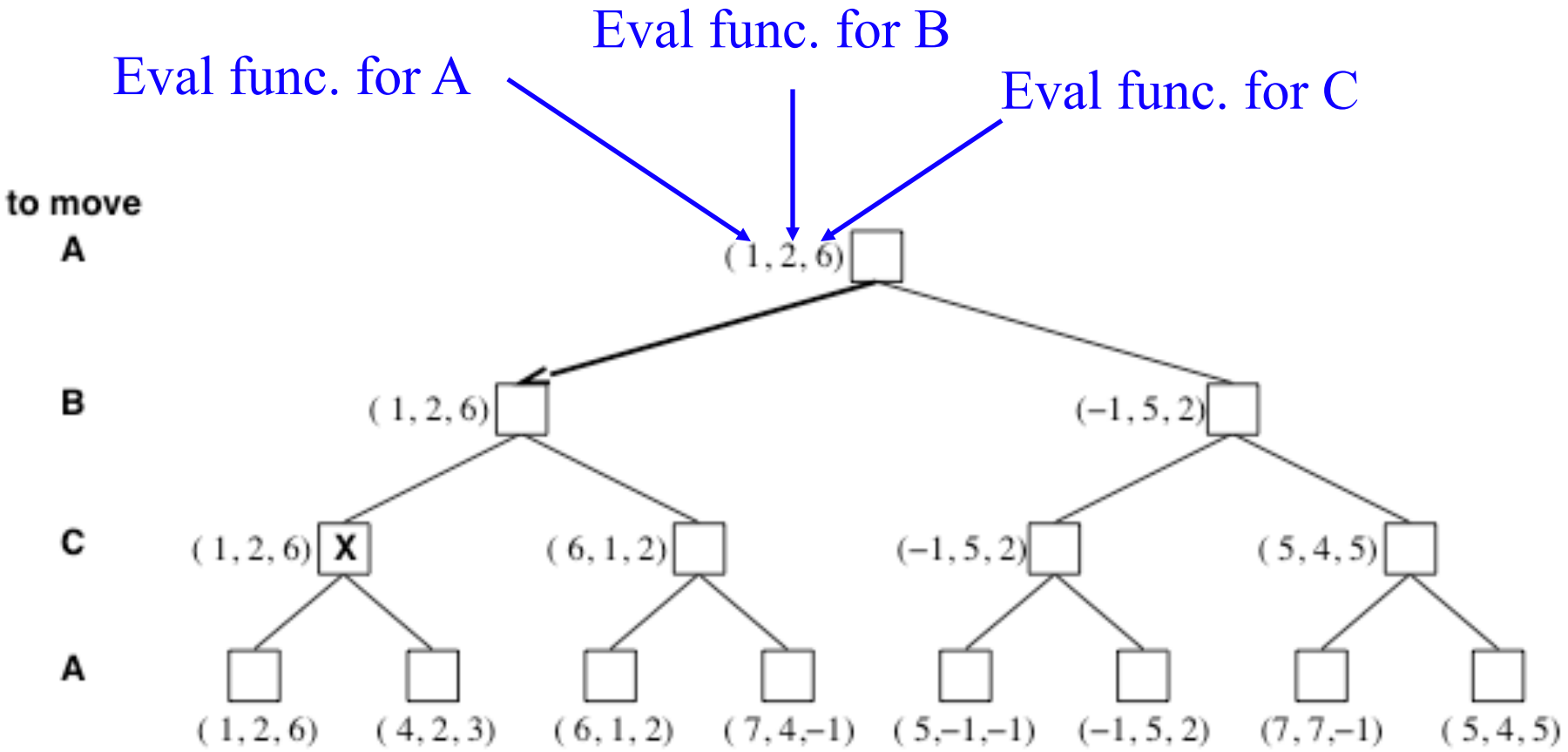


Alpha-beta pruning

Multiplayer games

- Games allow more than two players
- Solution:
  - Single minimax values become vectors
  - For example, 3 players: A, B, C
  - Values at each node: [V_A, V_B, V_C]
    - ✓ V_A: measure the utility of the state under in viewpoint of A
    - ✓ V_B: measure the utility of the state under in viewpoint of B
    - ✓ V_C: measure the utility of the state under in viewpoint of C
- Multiplayer → Making Alliances

Multiplayer games



Multiplayer games

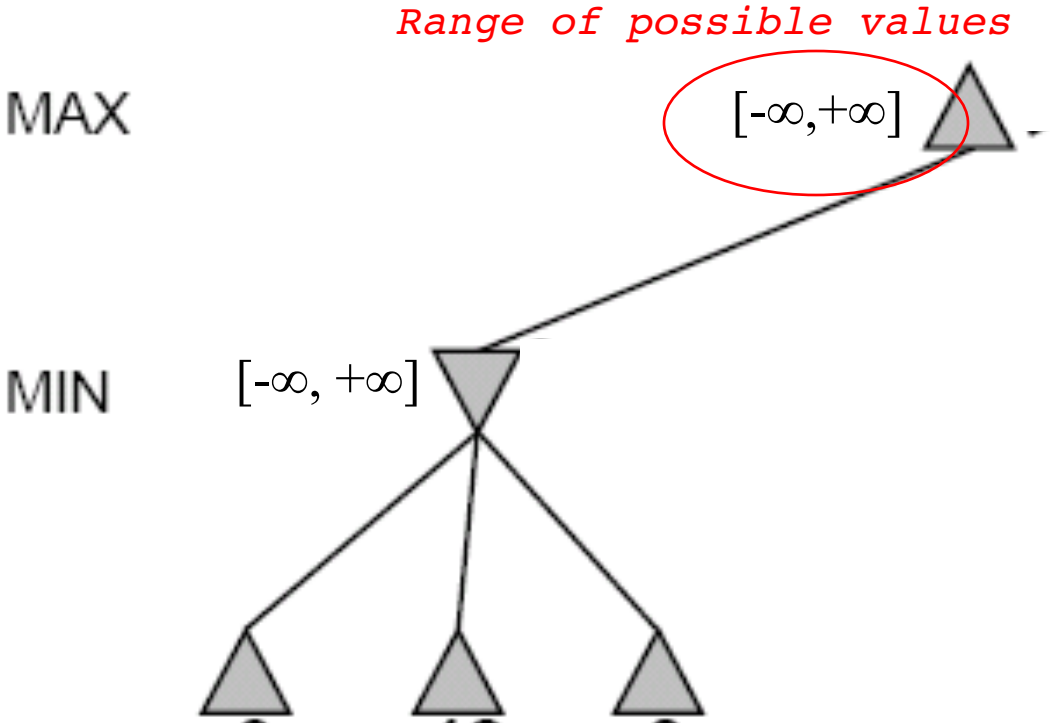
- Multiplayer → Strategy: Making Alliances, look like "Three Kingdoms" in Chinese history
  - A and B are in their weak points
  - C is in its stronger point
  - → It is a reasonable that A and B make a alliance and together attack C
  - …
  - However, as soon as C weakens the alliance is break down.

Problem of minimax search

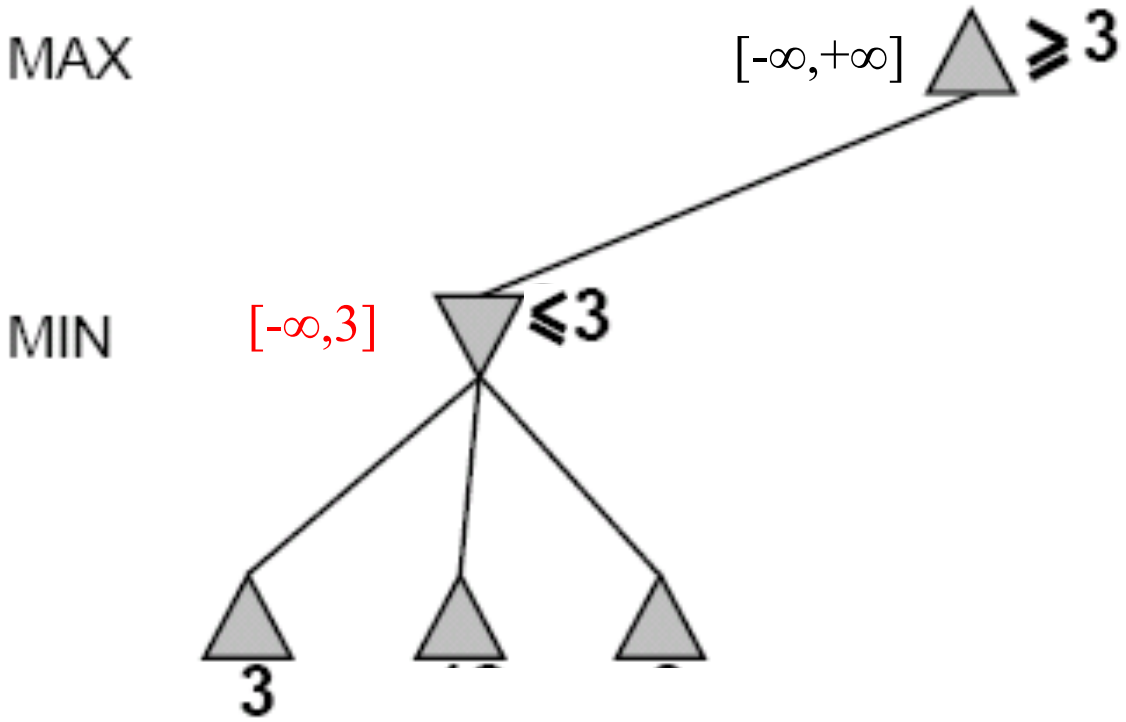
- Number of games states is exponential to the number of moves.
  - Solution: Do not examine every node
  - ==> Alpha-beta pruning
    - ✓ Alpha = value of best choice found so far at any choice point along the MAX path
    - ✓ Beta = value of best choice found so far at any choice point along the MIN path
- Revisit example …

Alpha-Beta Example

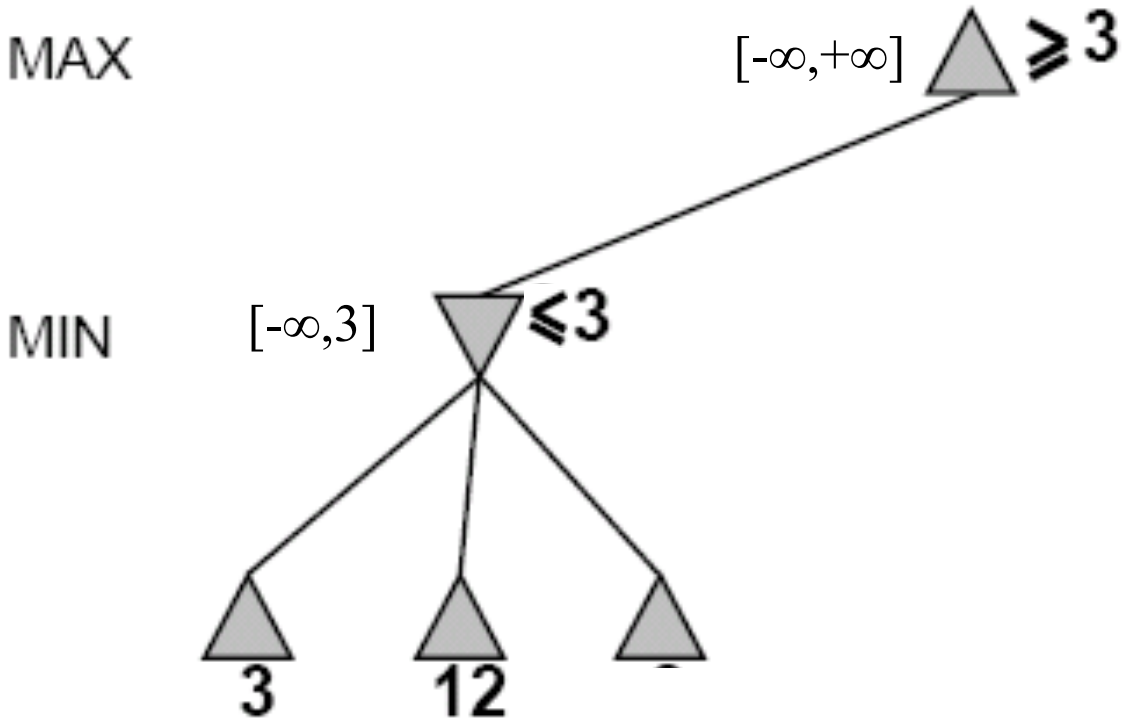
Do DF-search until first leaf



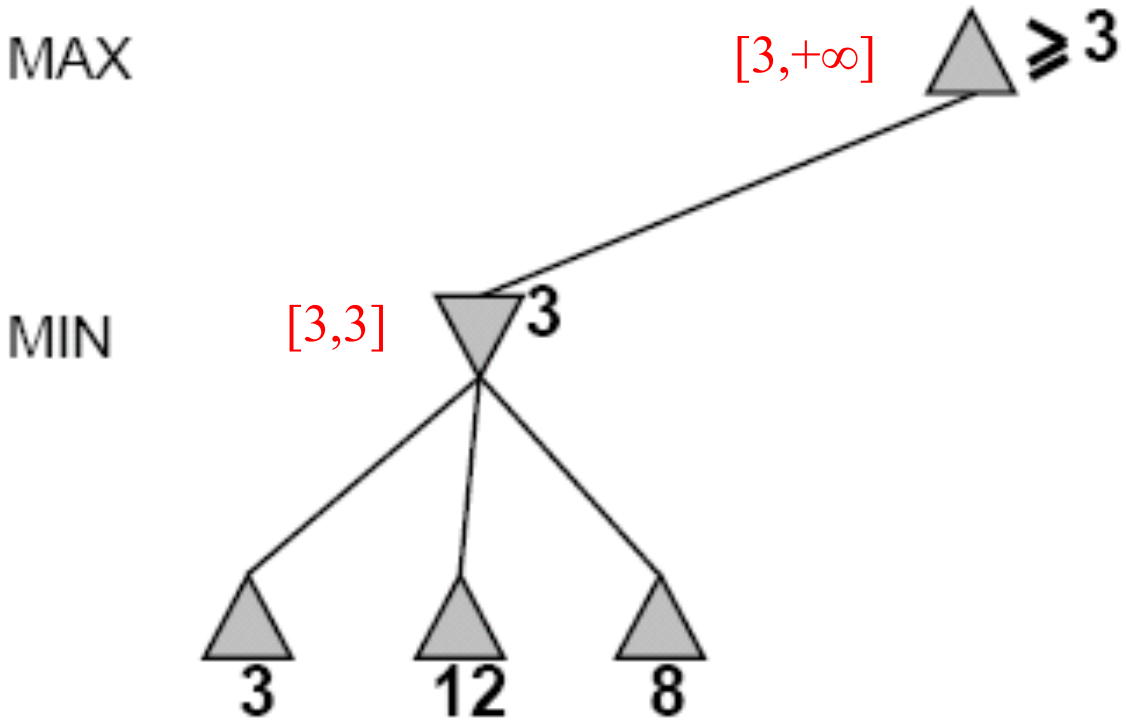
Alpha-Beta Example (continued)



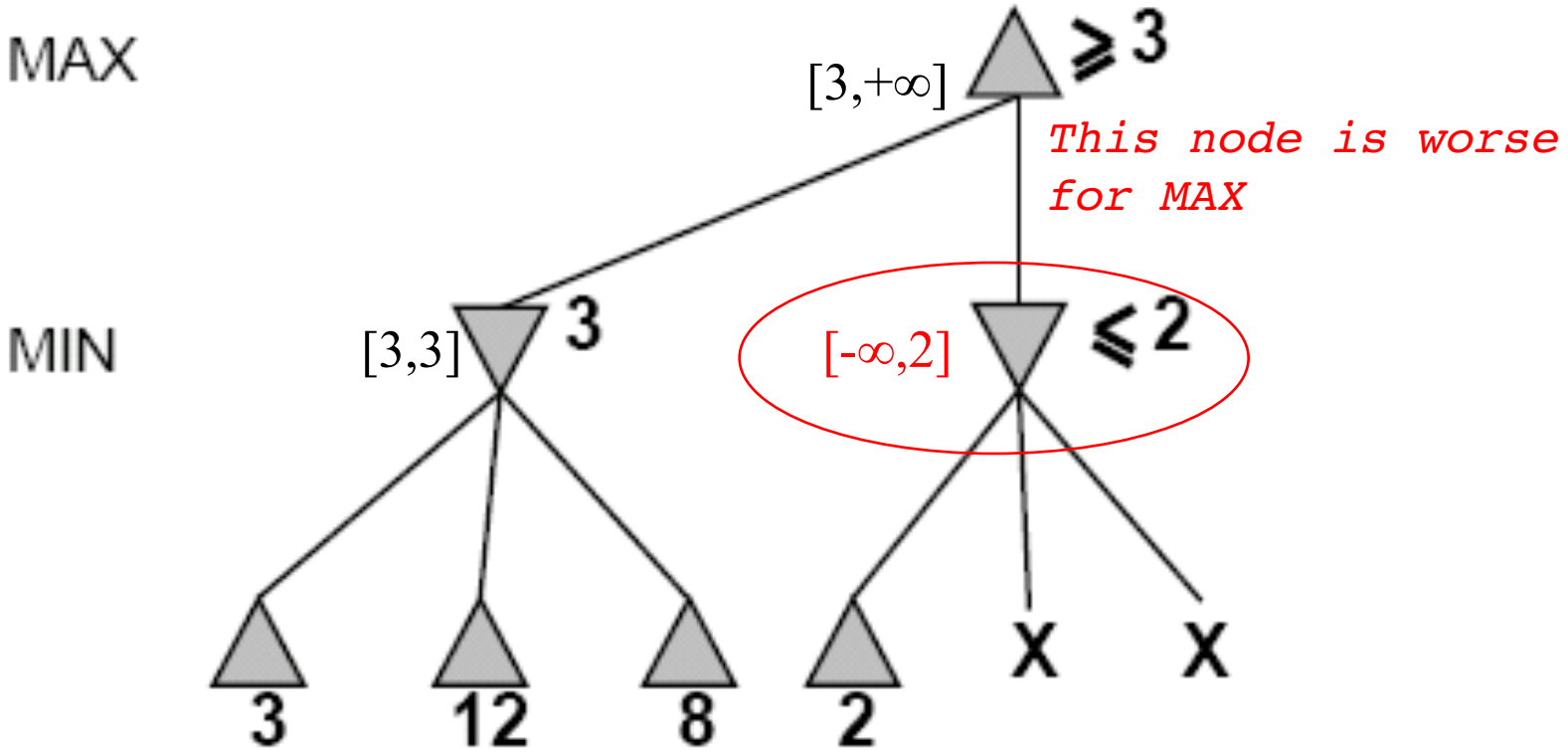
Alpha-Beta Example (continued)



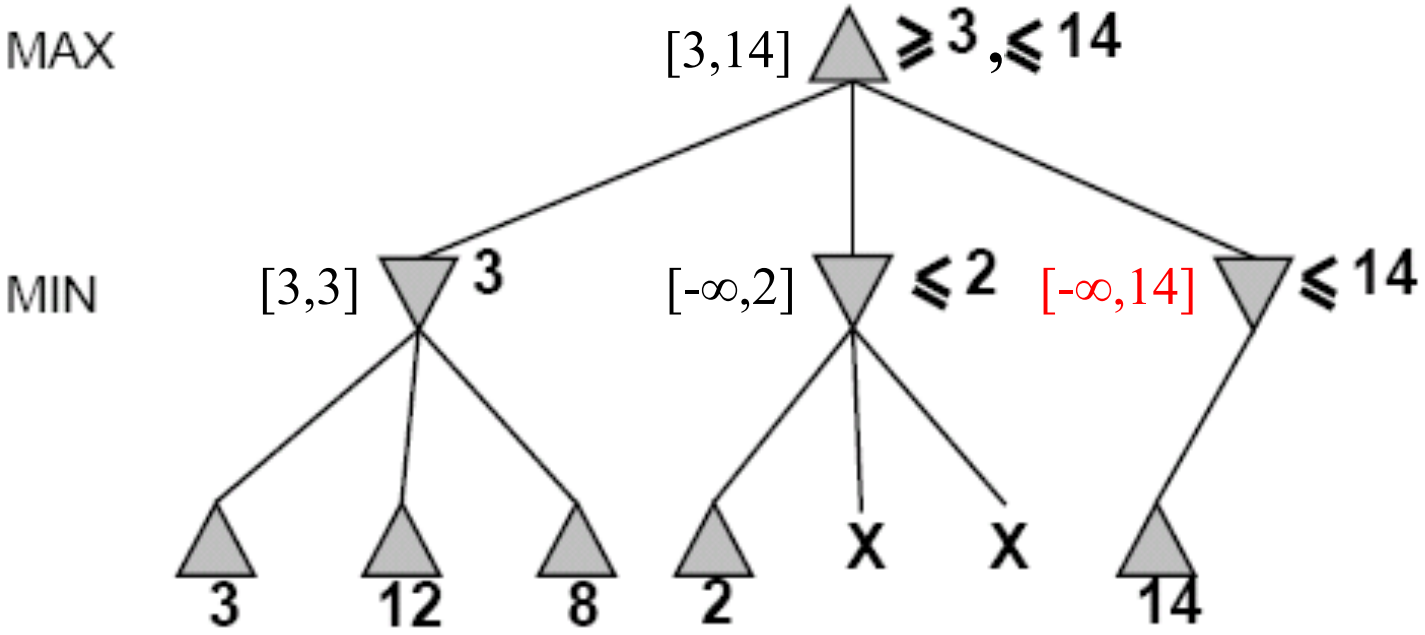
Alpha-Beta Example (continued)



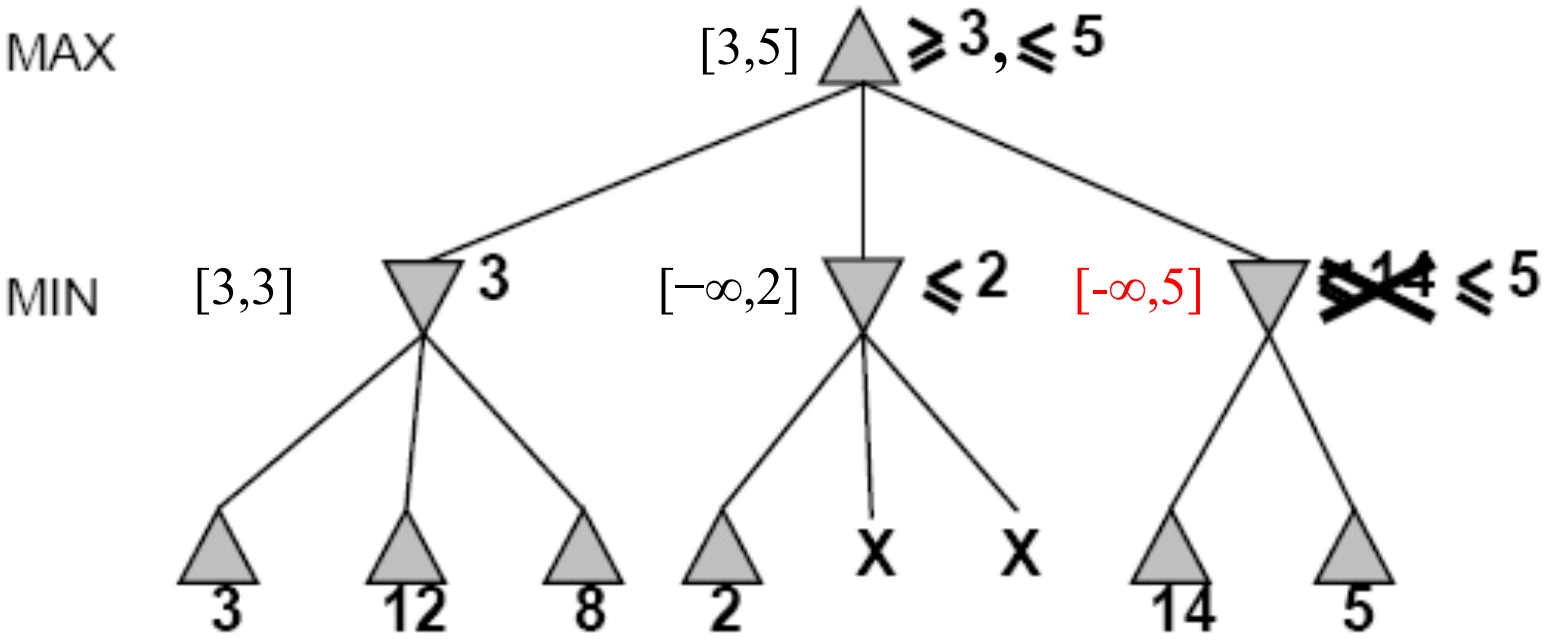
Alpha-Beta Example (continued)



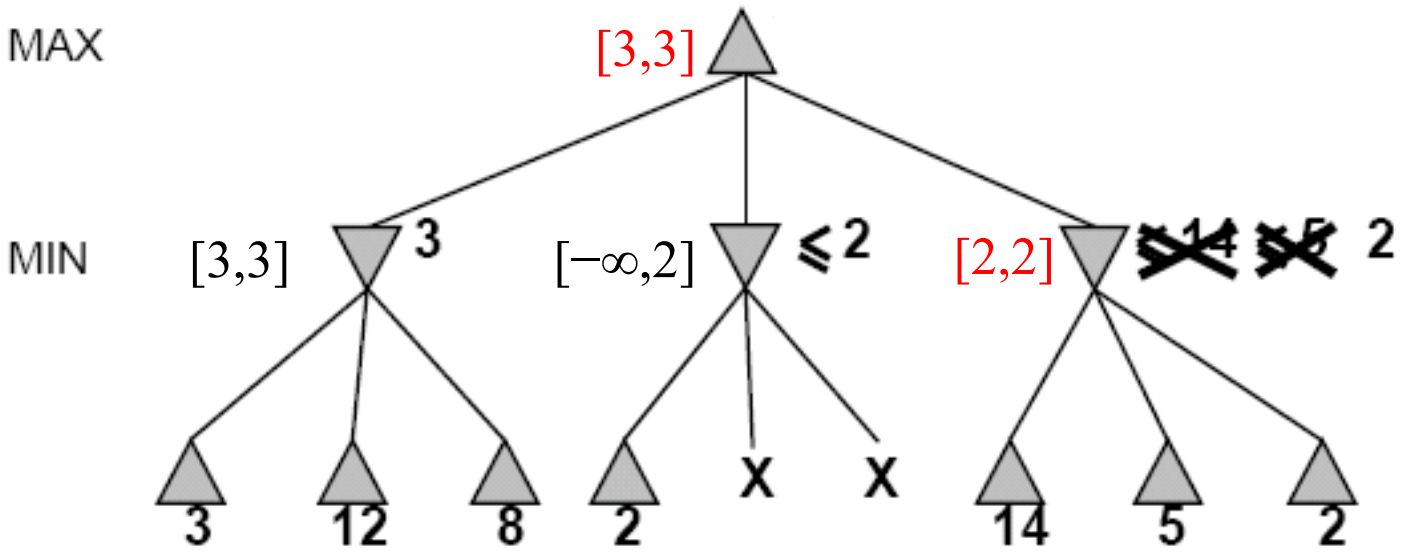
Alpha-Beta Example (continued)



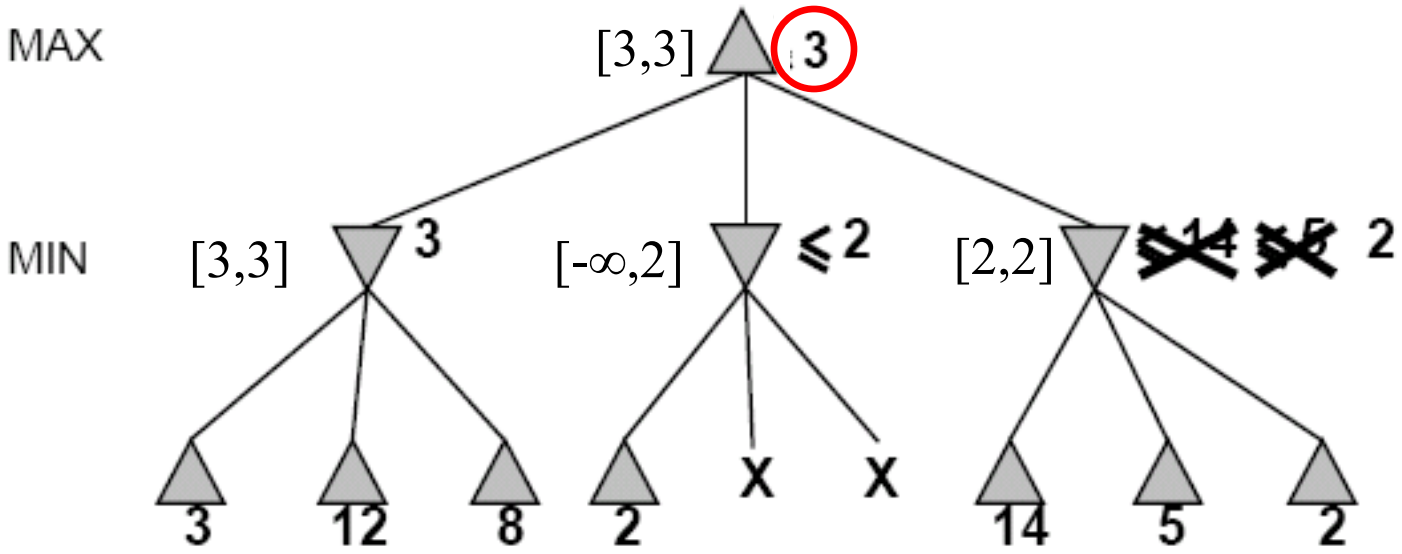
Alpha-Beta Example (continued)

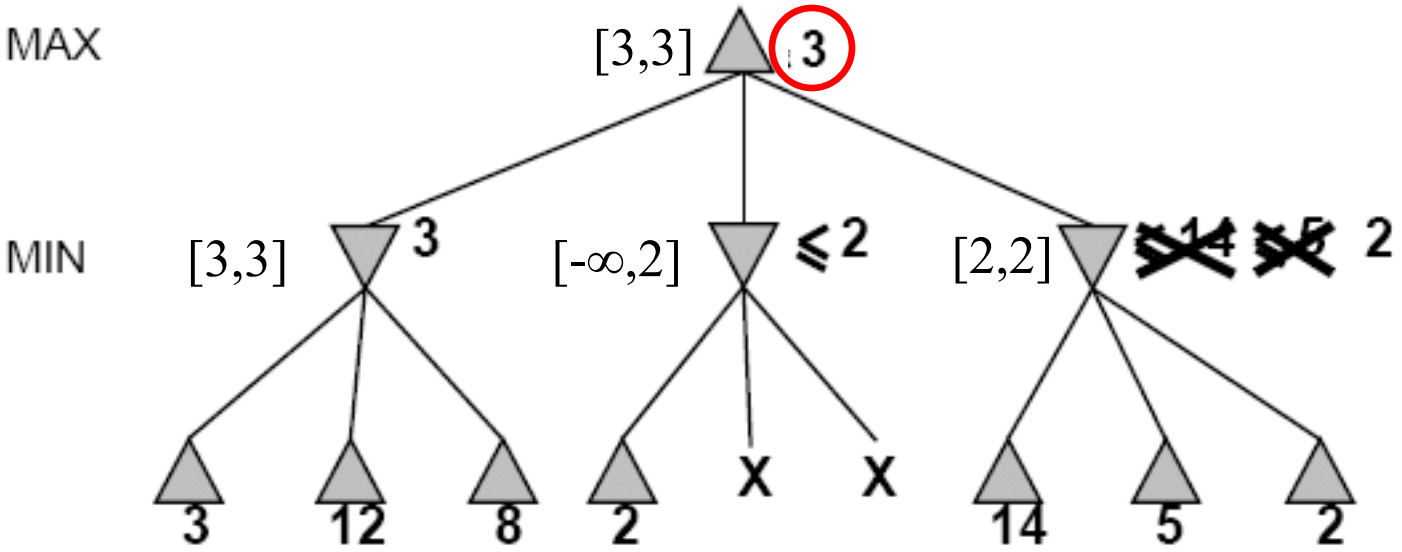


Alpha-Beta Example (continued)



Alpha-Beta Example (continued)





$\alpha =$ the value of the best (i.e., highest-value) choice we have found so far at any choice point along the path for MAX.

$\beta =$ the value of the best (i.e., lowest-value) choice we have found so far at any choice point along the path for MIN.

Alpha-Beta Algorithm

function ALPHA-BETA-SEARCH(*state*) **returns** *an action*
  inputs: *state*, current state in game
  $v \leftarrow$ MAX-VALUE(*state*, $-\infty$, $+\infty$)
  return the *action* in SUCCESSORS(*state*) with value *v*

---

function MAX-VALUE(*state*, α , β) **returns** *a utility value*
  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  $v \leftarrow -\infty$
  for *a,s* in SUCCESSORS(*state*) **do**
    $v \leftarrow$ MAX(*v*,MIN-VALUE(*s*, α , β))
    if $v \geq \beta$ **then return** v
    $\alpha \leftarrow$ MAX(α ,*v*)
  return v

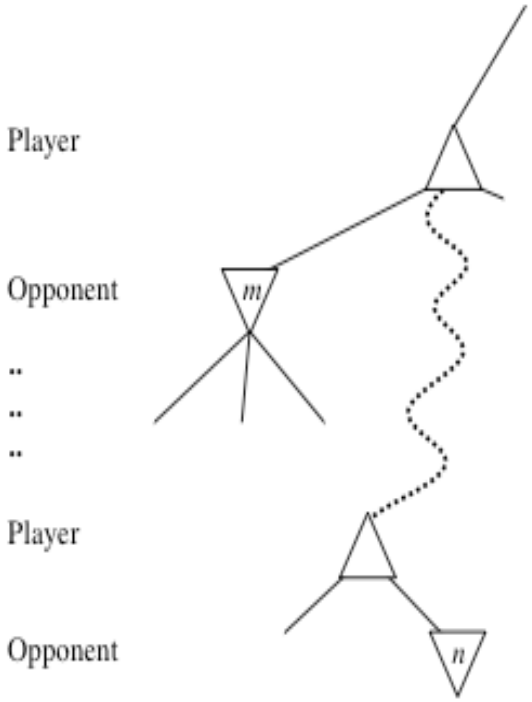
Alpha-Beta Algorithm

---

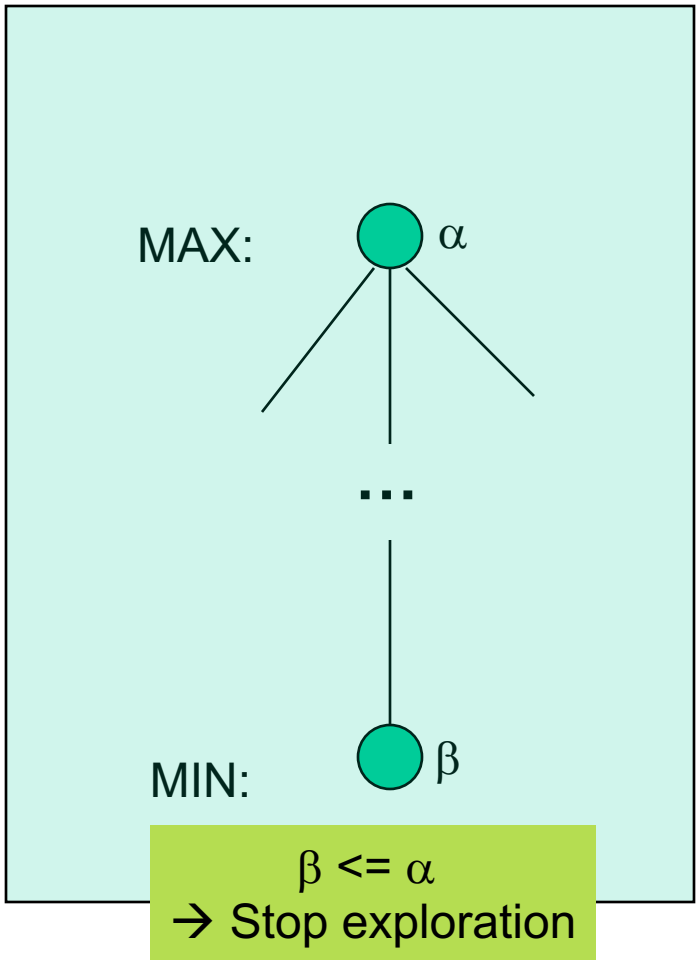
function MIN-VALUE(*state*, α, β) **returns** *a utility value*
  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  $v \leftarrow +\infty$
  for *a,s* in SUCCESSORS(*state*) **do**
    $v \leftarrow$ MIN(v,MAX-VALUE(*s*, α, β))
   if $v \leq \alpha$ **then return** v
    $\beta \leftarrow$ MIN(β,v)
  return v

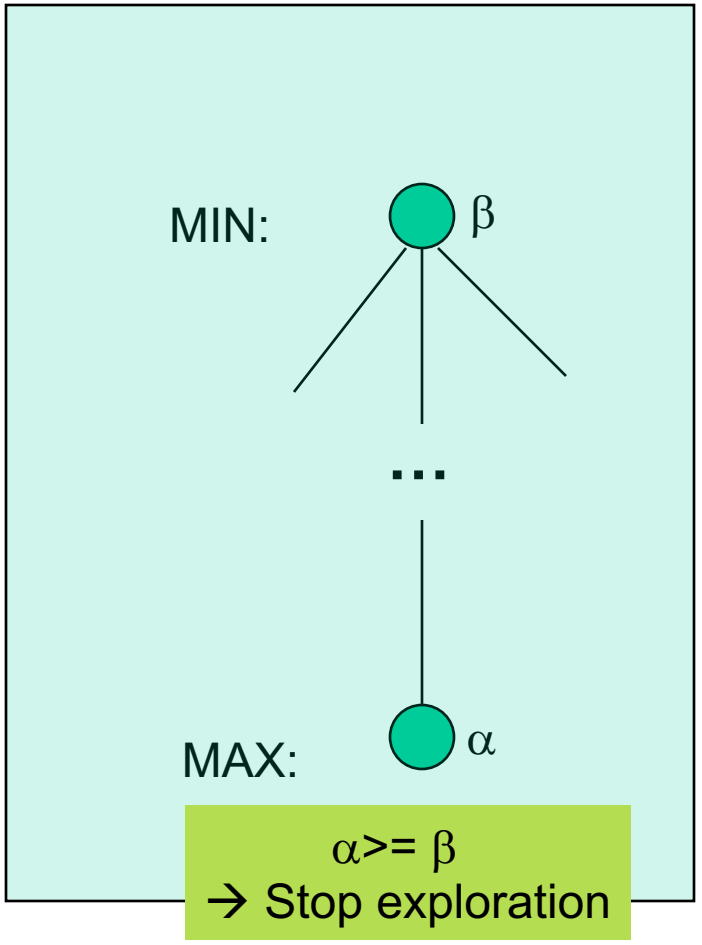
General alpha-beta pruning

- Consider a node *n* somewhere in the tree
- If player has a better choice at
  - Parent node of n
  - Or any choice point further up
- *n* will **never** be reached in actual play.
- Hence when enough is known about *n*, it can be pruned.

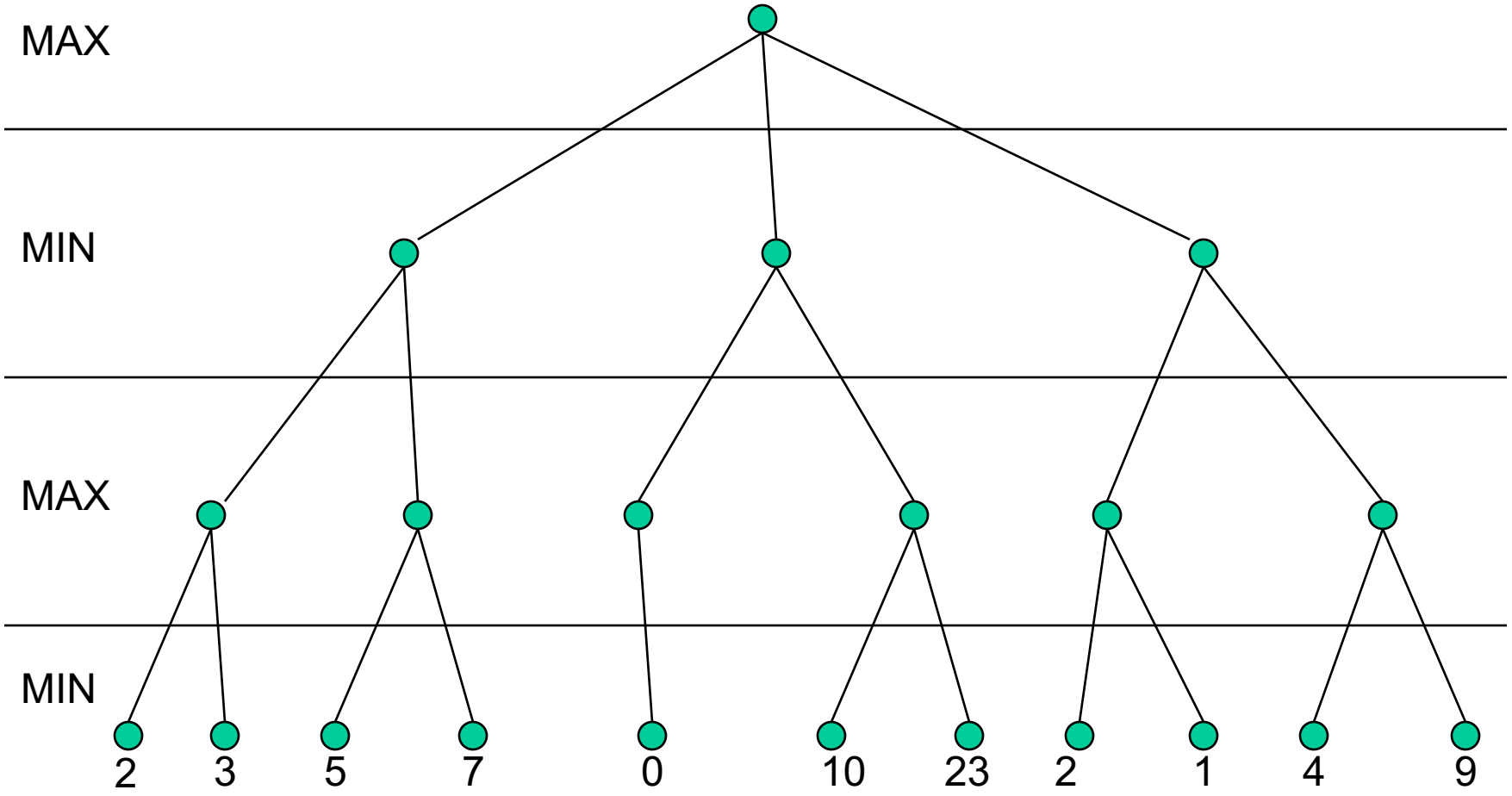


Alpha-Beta Pruning



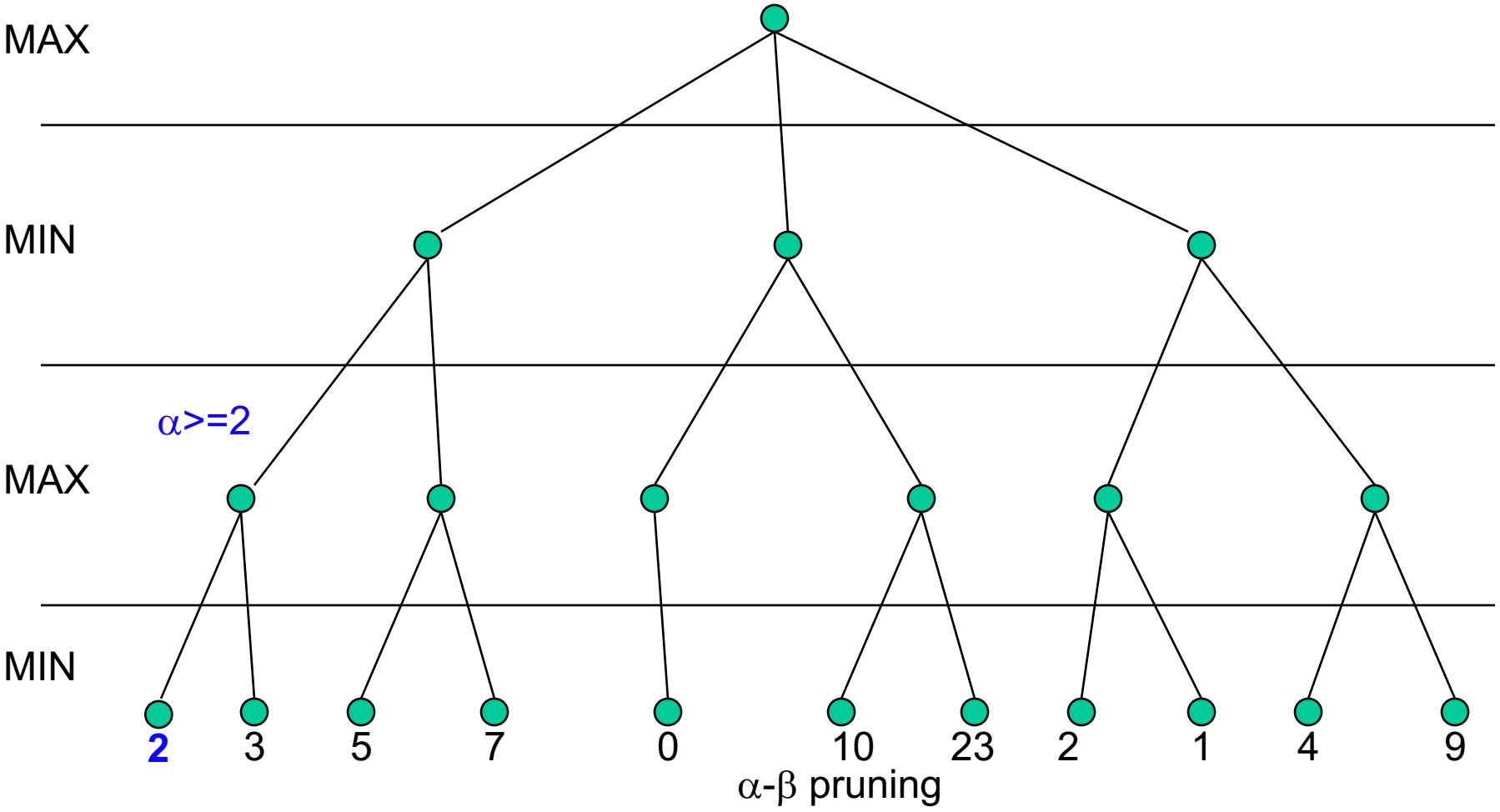


Alpha-Beta Example

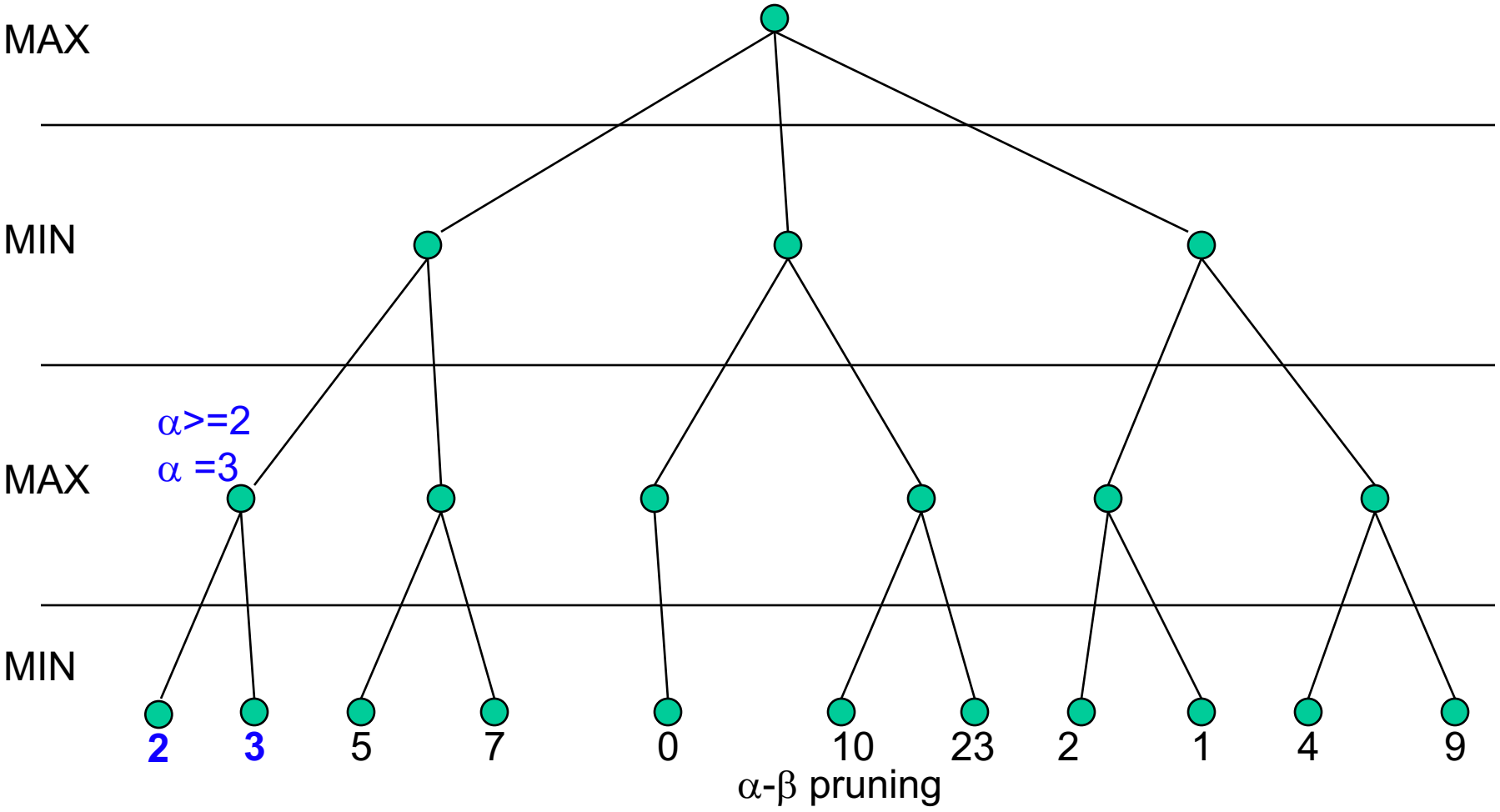


Suppose that the search tree has the form as shown in the figure

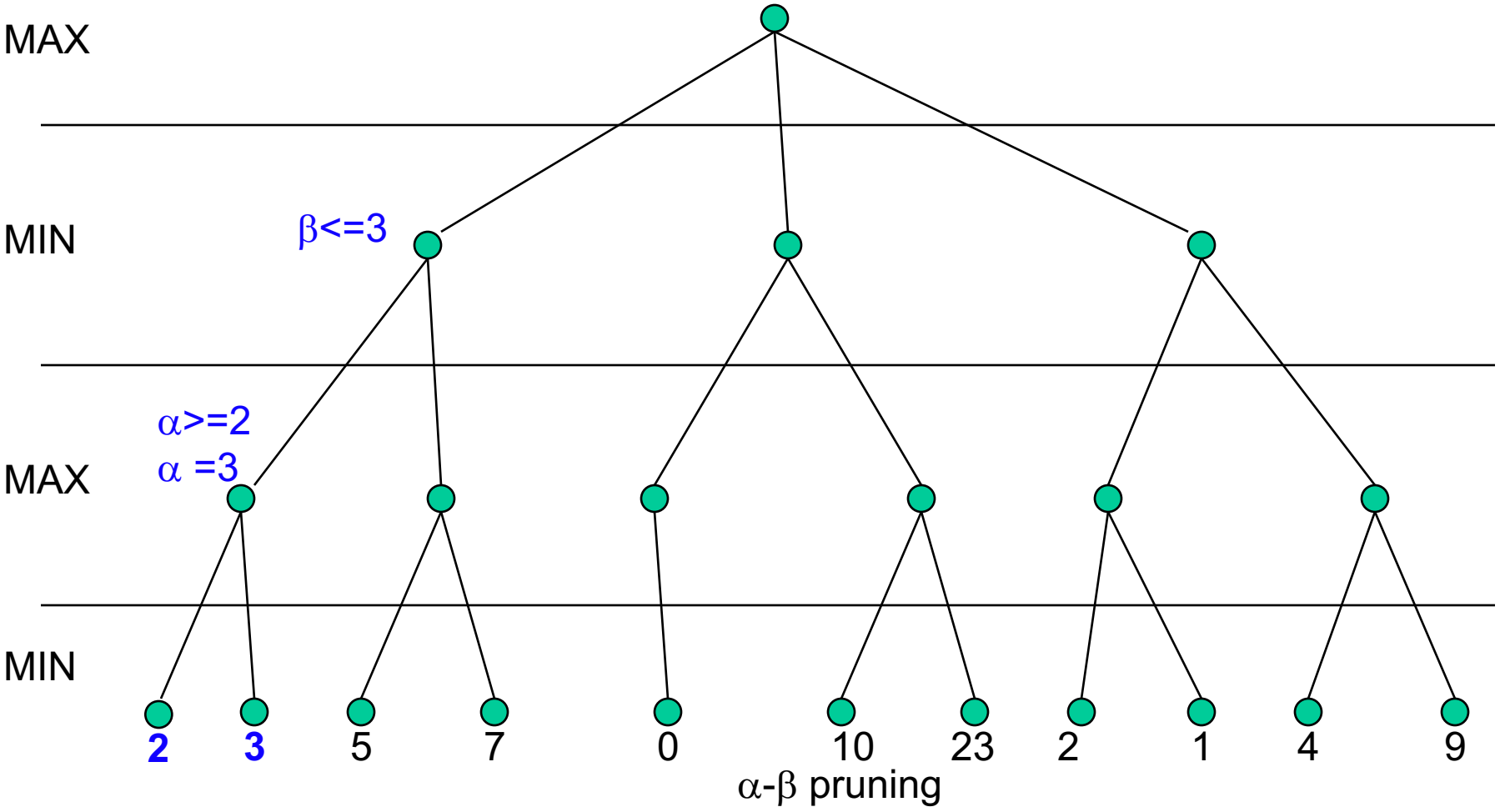
Alpha-Beta Example



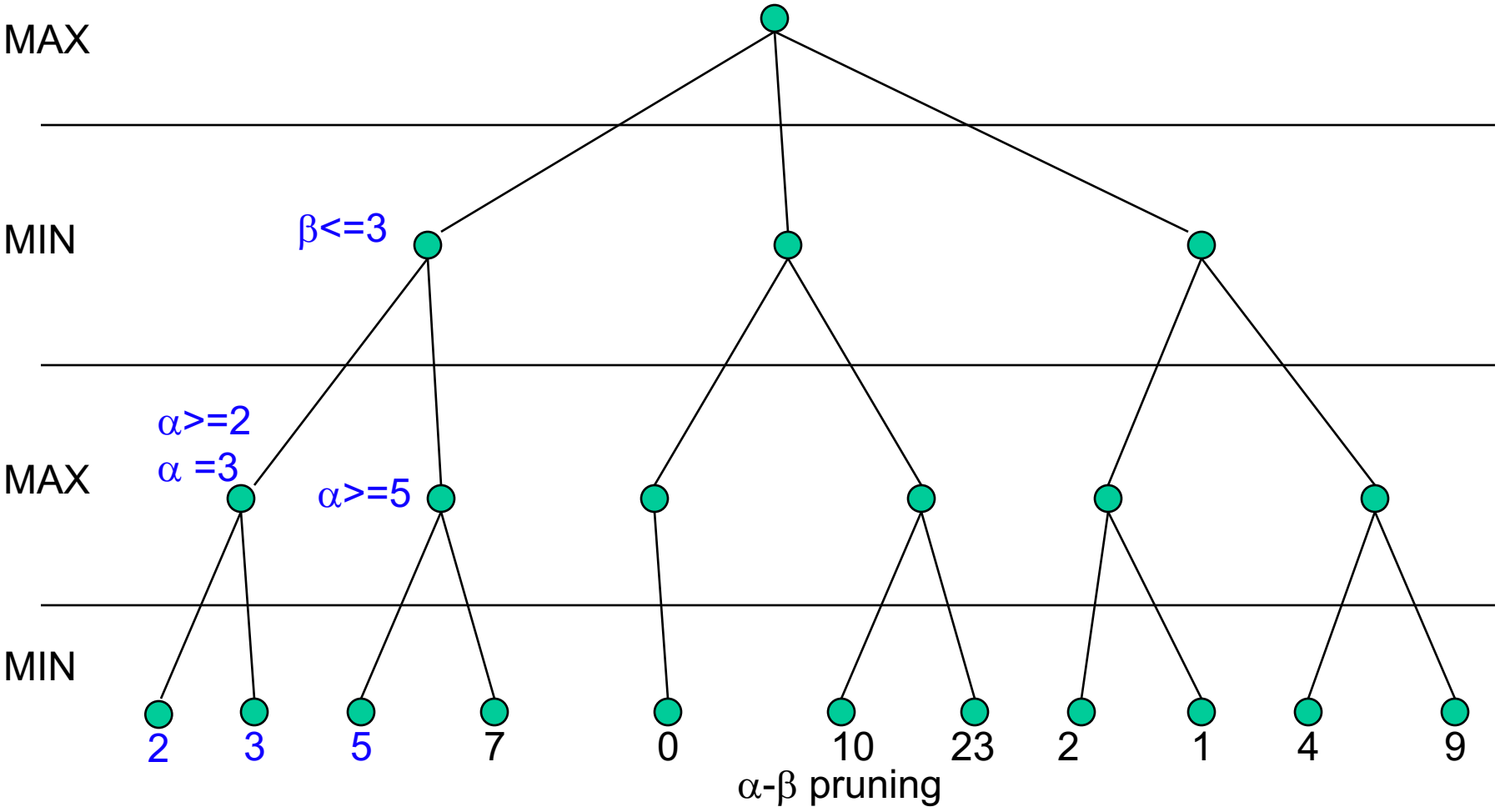
Alpha-Beta Example



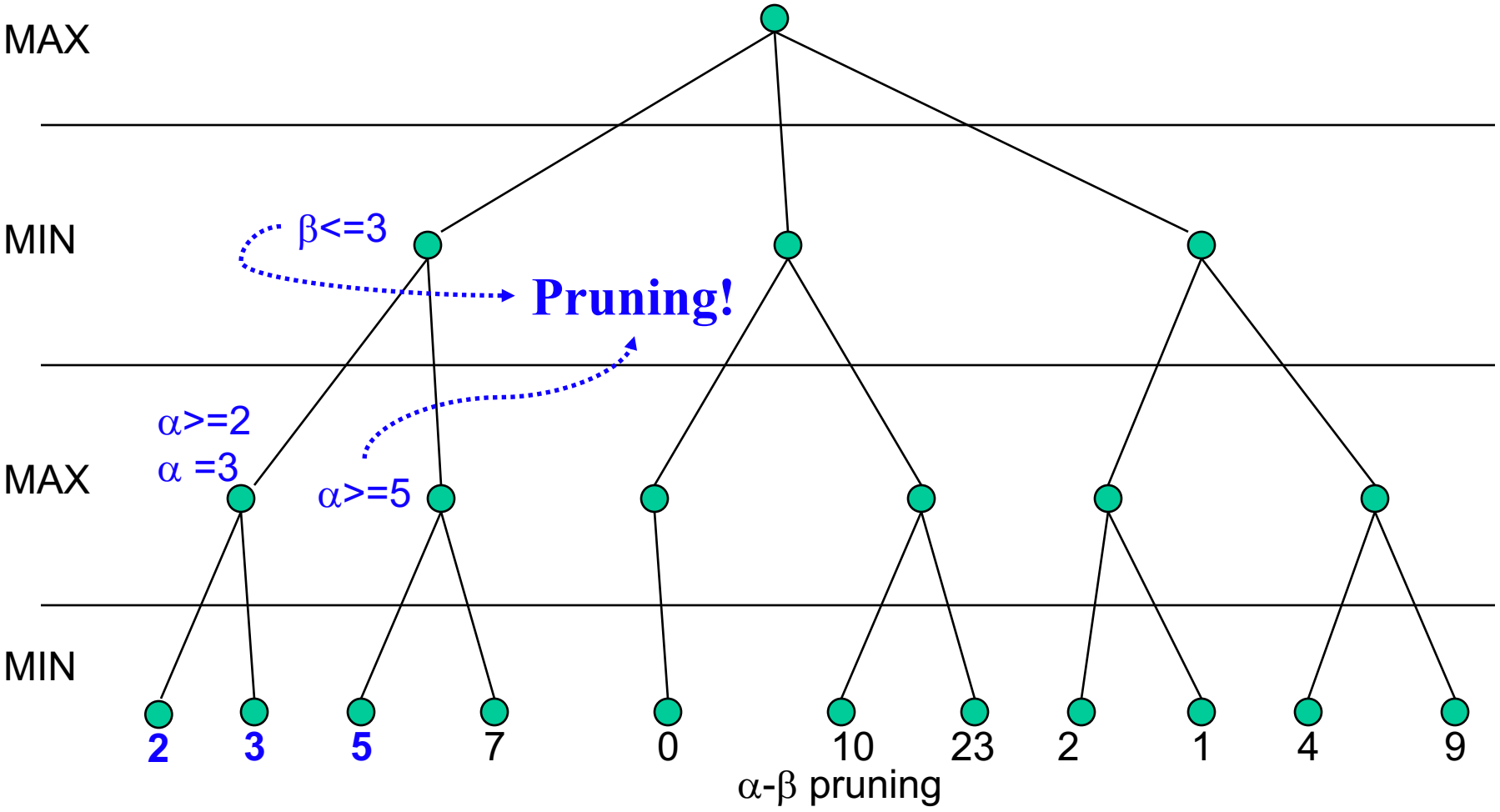
Alpha-Beta Example



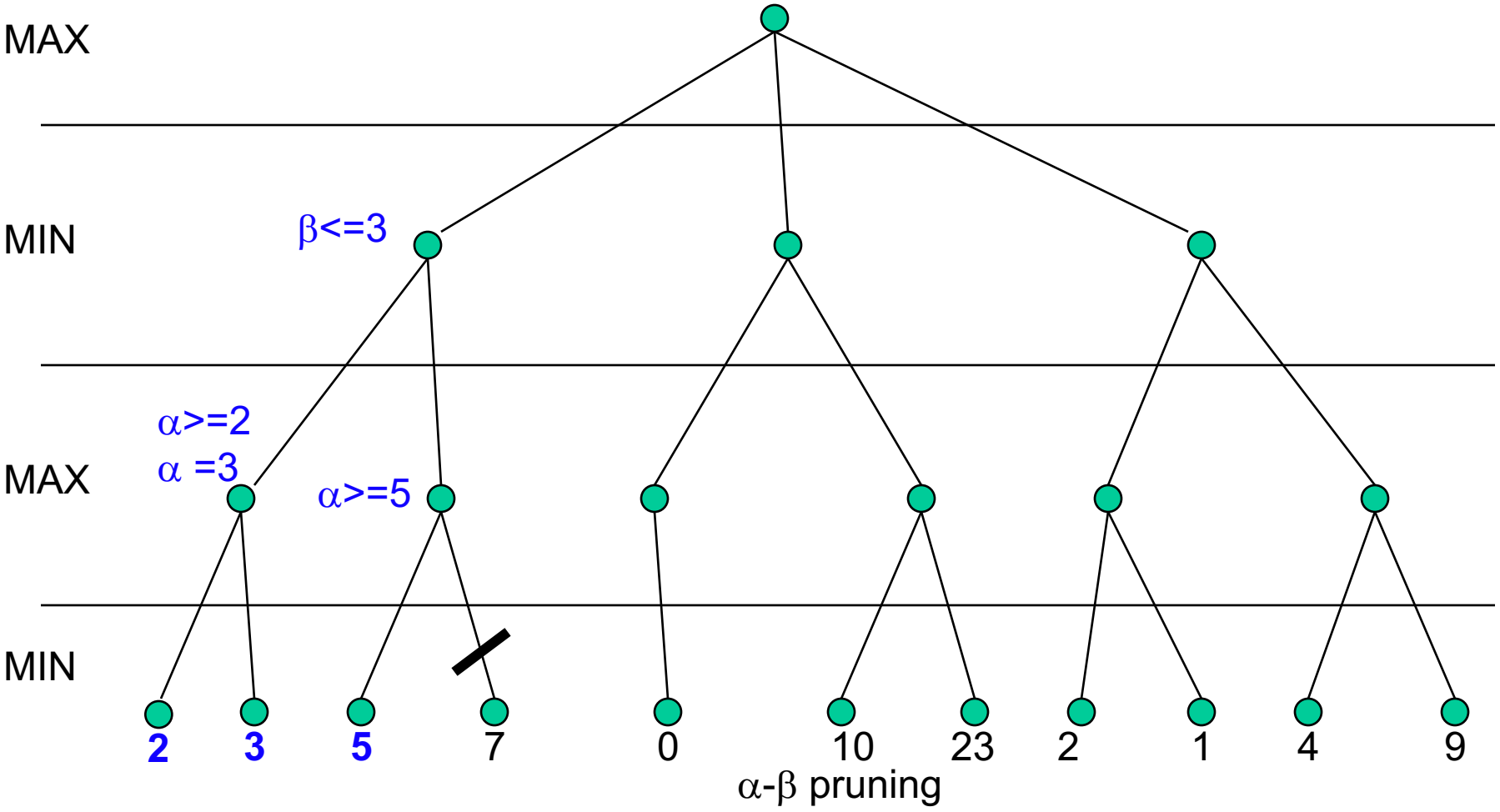
Alpha-Beta Example



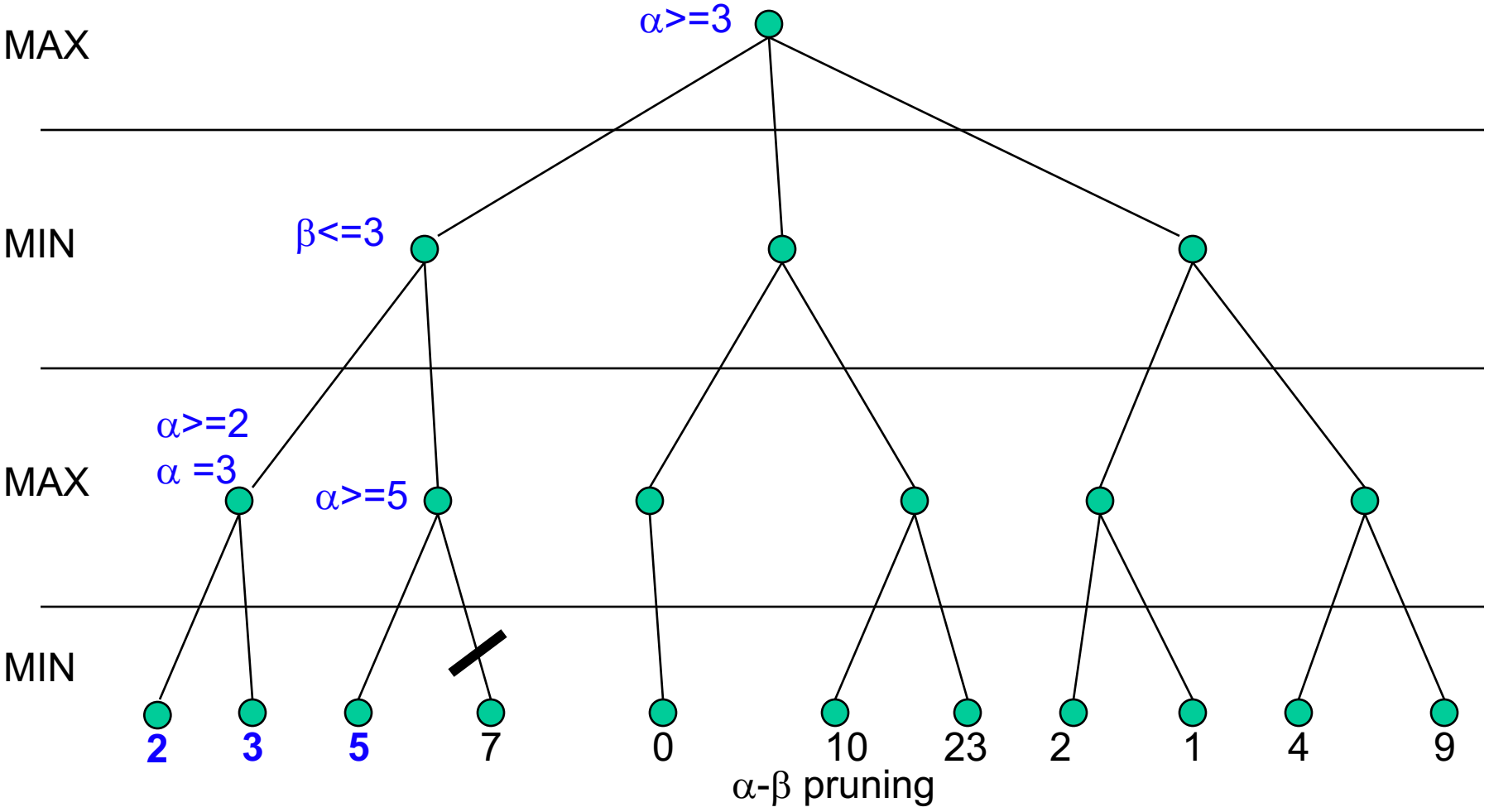
Alpha-Beta Example



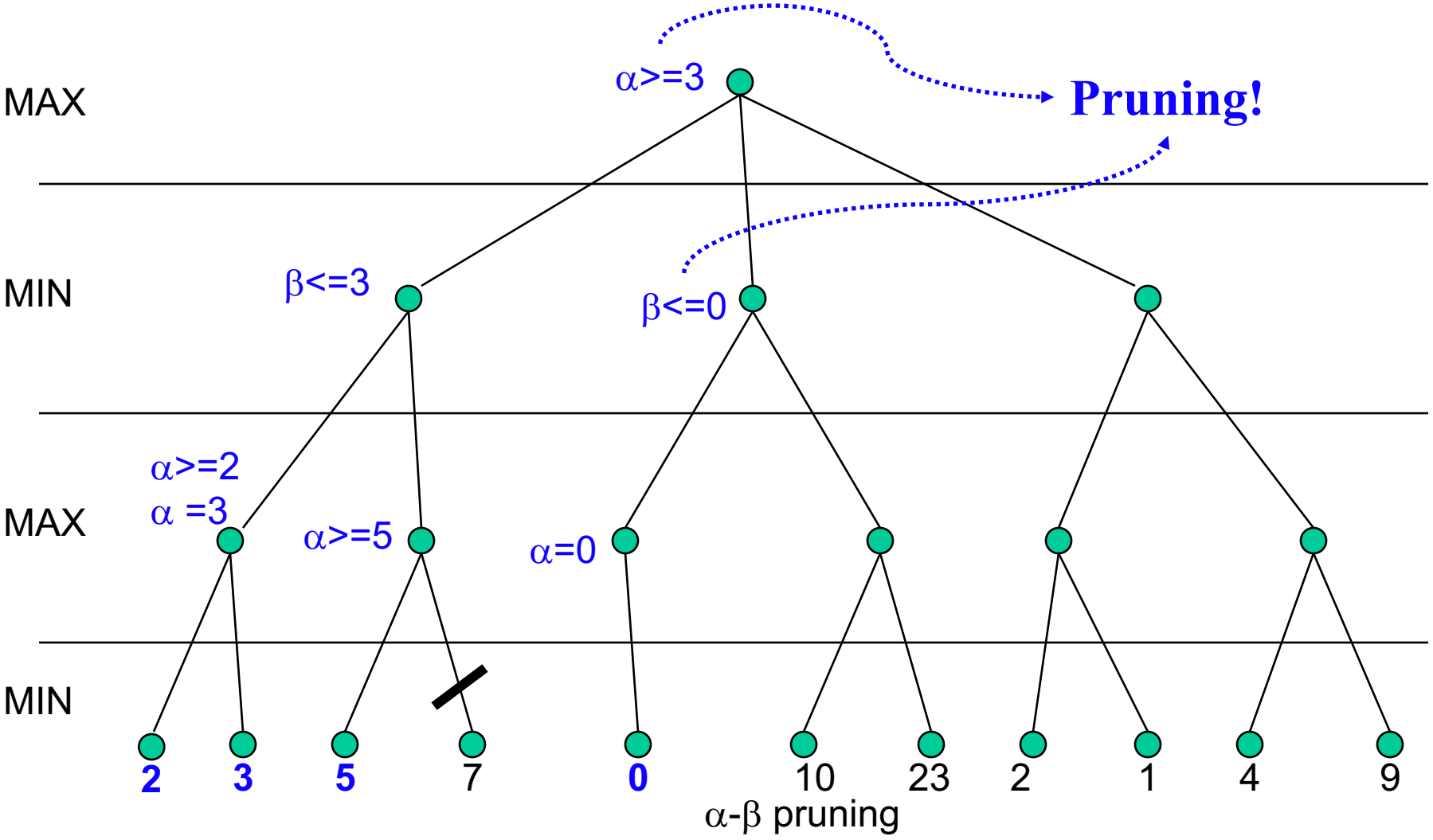
Alpha-Beta Example



Alpha-Beta Example

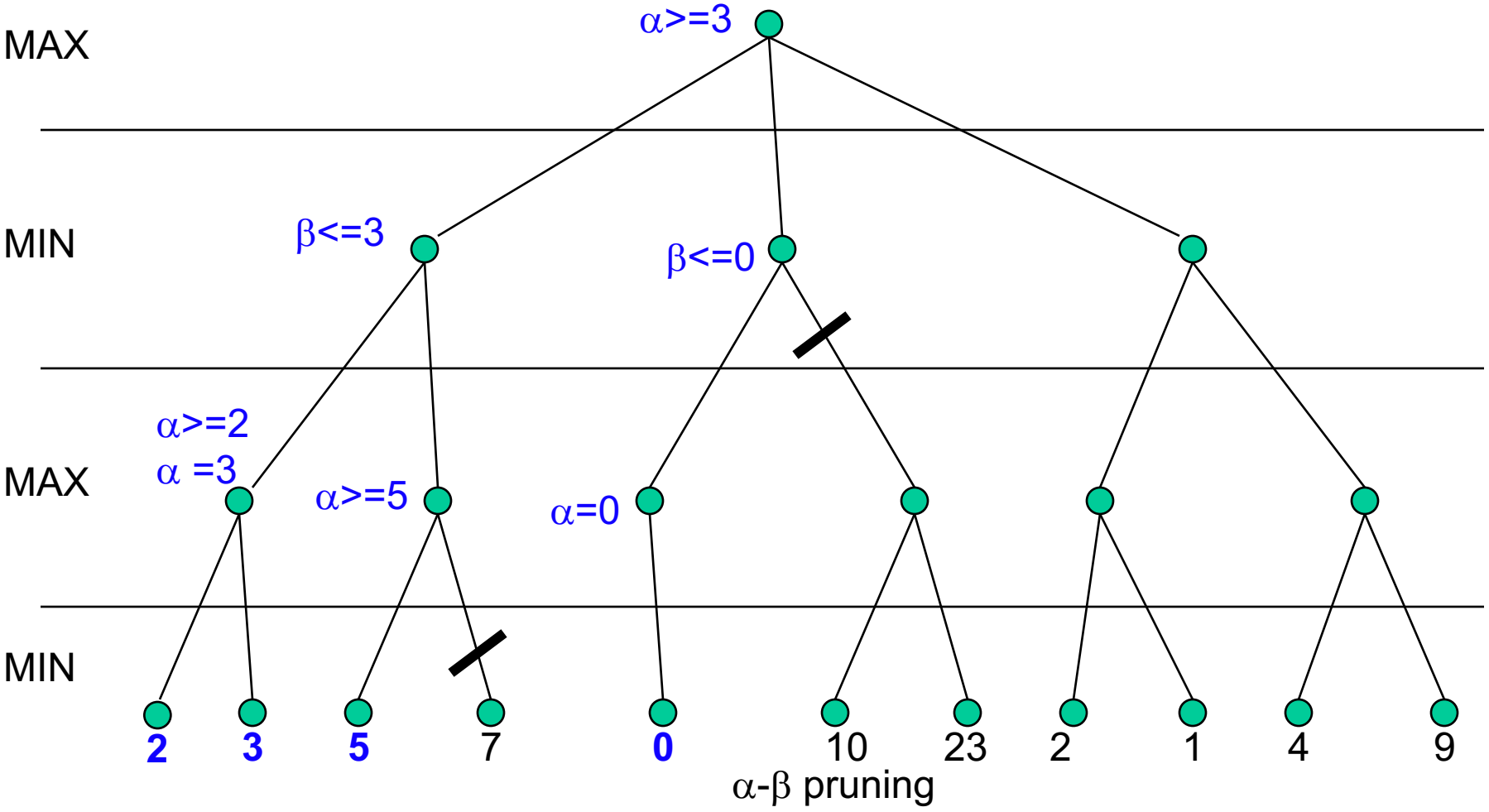


Alpha-Beta Example

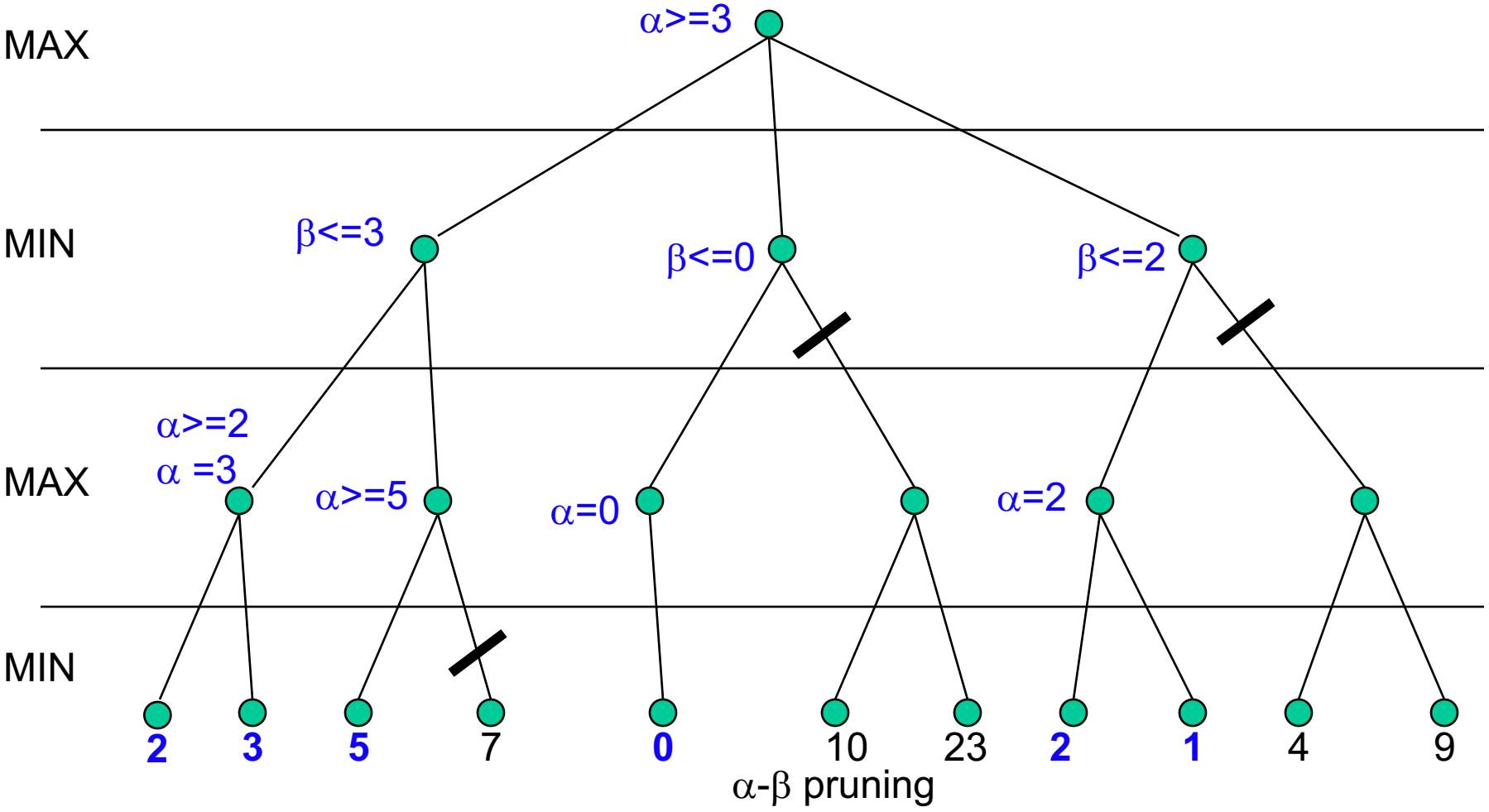


α-β pruning

Alpha-Beta Example



Alpha-Beta Example



Final Comments about Alpha-Beta Pruning

- Pruning does not affect final results
- Entire subtrees can be pruned.
- Good move *ordering* improves effectiveness of pruning
- With "perfect ordering," time complexity is $O(b^{m/2})$
  - Branching factor of sqrt(b) !!
  - Alpha-beta pruning can look twice as far as minimax in the same amount of time
- Repeated states are again possible.
  - Store them in memory = transposition table

Imperfect information

Games of imperfect information

- Minimax and alpha-beta pruning require too much leaf-node evaluations.
- May be impractical within a reasonable amount of time.
- SHANNON (1950):
  - Cut off search earlier (replace TERMINAL-TEST by CUTOFF-TEST)
  - Apply heuristic evaluation function EVAL (replacing utility function of alpha-beta)

Cutting off search

- Change:
  - **if** TERMINAL-TEST(*state*) **then return** UTILITY(*state*)

  into
  - **if** CUTOFF-TEST(*state,depth*) **then return** EVAL(*state*)

- Introduces a fixed-depth limit *depth*
  - Is selected so that the amount of time will not exceed what the rules of the game allow.
- When cuttoff occurs, the evaluation is performed.

Alpha-Beta Algorithm, Rivisted

function ALPHA-BETA-SEARCH(*state*) **returns** *an action*
  inputs: *state*, current state in game
  $v \leftarrow$ MAX-VALUE(*state*, $-\infty$, $+\infty$)
  return the *action* in SUCCESSORS(*state*) with value *v*

---

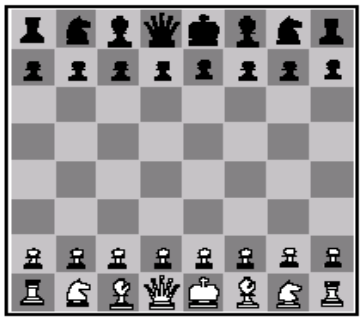
function MAX-VALUE(*state*, α , β) **returns** *a utility value*
  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  $v \leftarrow -\infty$
  for *a*,*s* in SUCCESSORS(*state*) **do**
    $v \leftarrow$ MAX(v,MIN-VALUE(*s*, α , β))
   if $v \geq \beta$ **then return** v
   $\alpha \leftarrow$ MAX(α ,v)
  return v

if CUTOFF-TEST(*state*,*depth*)
then return EVAL(*state*)

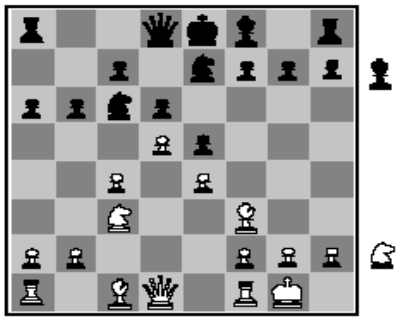
Heuristic EVAL

- Idea: produce an estimate of the expected utility of the game from a given position.
- Performance depends on quality of EVAL.
- Requirements:
  - EVAL should order terminal-nodes in the same way as UTILITY.
  - Computation may not take too long.
  - For non-terminal states the EVAL should be strongly correlated with the actual chance of winning.
- Only useful for quiescent (no wild swings in value in near future) states

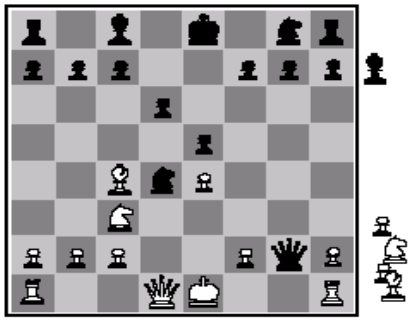
Heuristic EVAL example



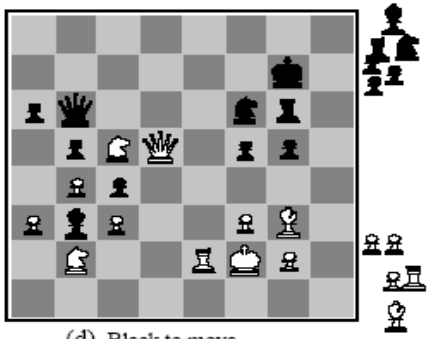
(a) White to move
Fairly even



(b) Black to move
White slightly better



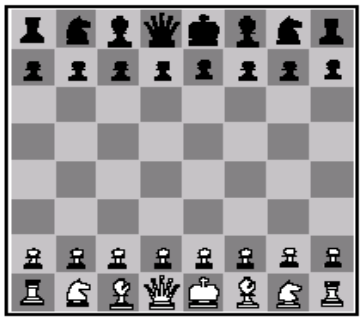
(c) White to move
Black winning



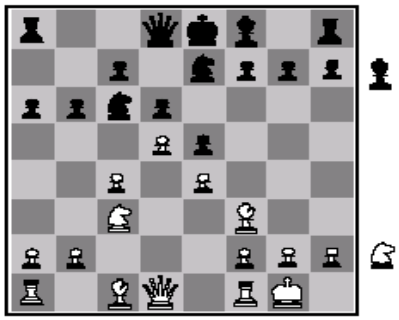
(d) Black to move
White about to lose

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

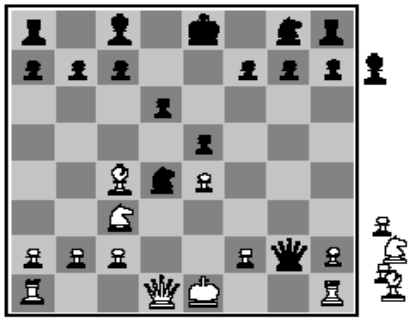
Heuristic EVAL example



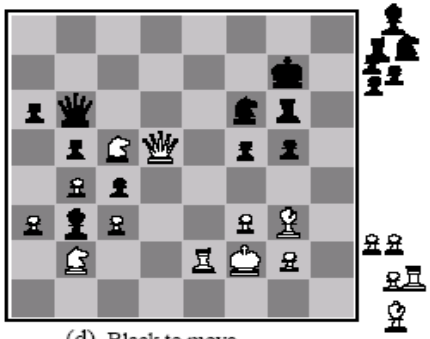
(a) White to move
Fairly even



(b) Black to move
White slightly better



(c) White to move
Black winning

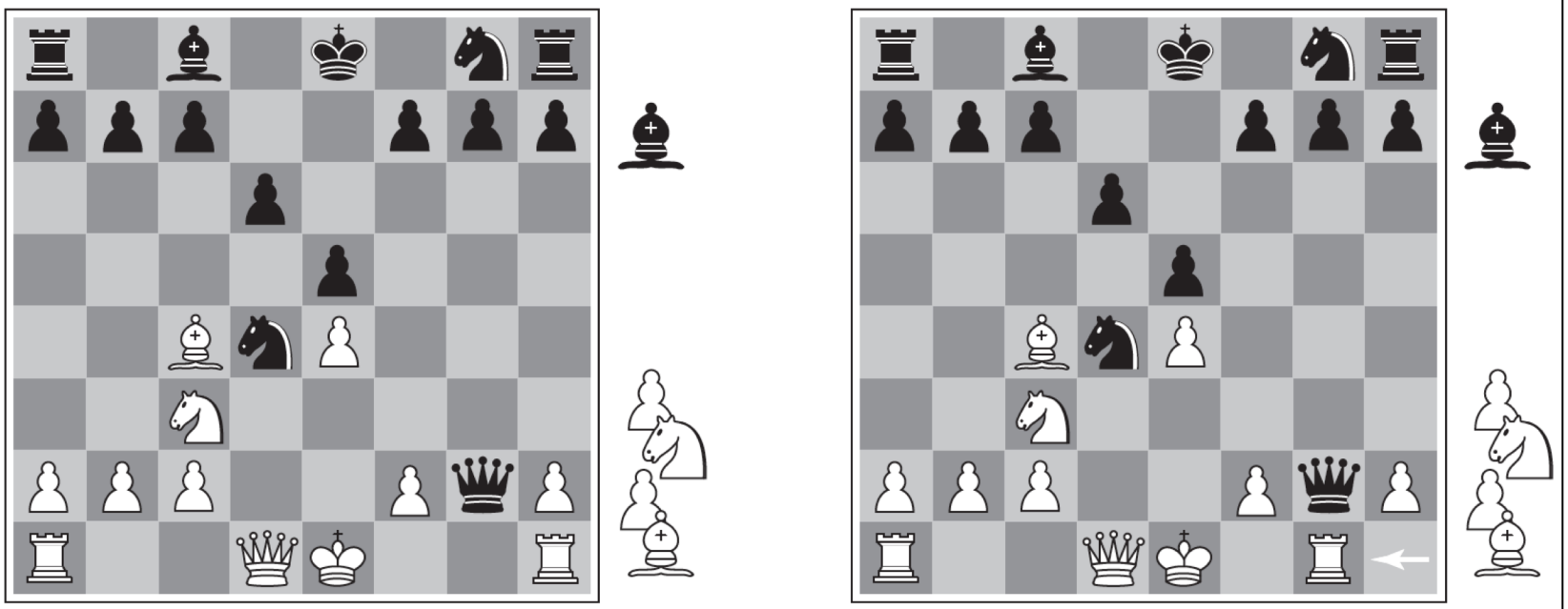


(d) Black to move
White about to lose

Addition assumes independence

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

Heuristic difficulties

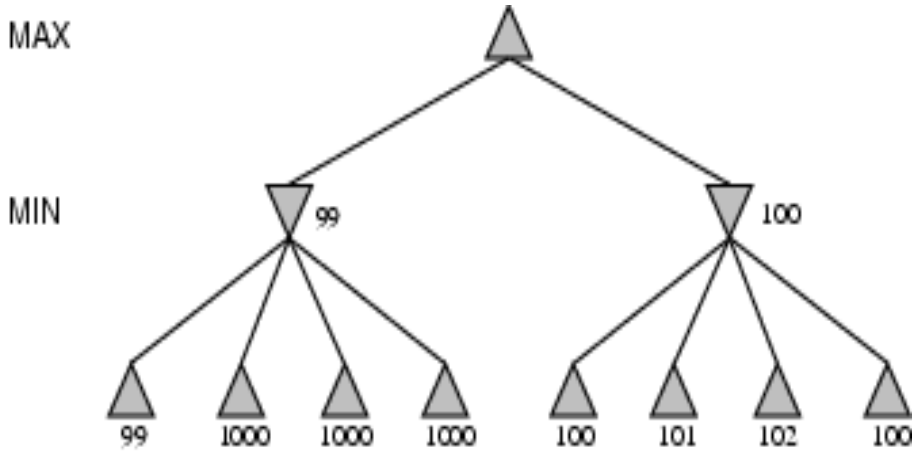


(a) White to move (b) White to move

Figure 5.8 Two chess positions that differ only in the position of the rook at lower right. In (a), Black has an advantage of a knight and two pawns, which should be enough to win the game. In (b), White will capture the queen, giving it an advantage that should be strong enough to win.

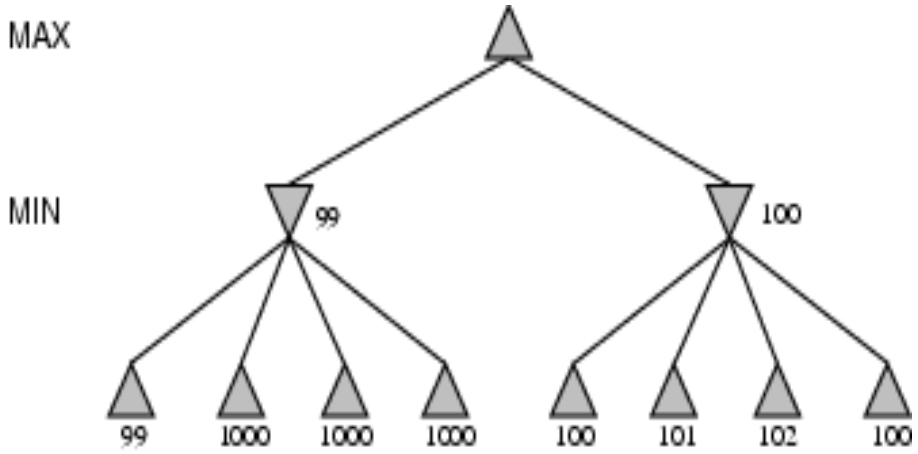
Discussion

- Examine section on state-of-the-art games yourself
- Minimax assumes right tree is better than left, yet …
  - Return probability distribution over possible values
  - Yet expensive calculation



Discussion

- Utility of node expansion
  - Only expand those nodes which lead to significanlty better moves
- Both suggestions require meta-reasoning



Summary

- Games are fun (and dangerous)
- They illustrate several important points about AI
  - Perfection is unattainable -> approximation
  - Good idea what to think about
  - Uncertainty constrains the assignment of values to states
- Games are to AI as grand prix racing is to automobile design.