CS 540 Introduction to Artificial Intelligence
Games II

University of Wisconsin-Madison

Spring 2023

Outline

Homeworks:

- Homework 9 due Thursday April 27
- Homework 10 due Thursday May 4

Course Evaluation:

Class roadmap:

Thursday, April 20	Games II
Tuesday, April 25	Reinforcement Learning I
Thursday, April 27	Reinforcement Learning I
Tuesday, May 2	Review of RL + Games
Thursday, May 4	Ethics and Trust in AI

Key Ideas in Games

Key Ideas in Games

Defining Games

Key Ideas in Games

Defining Games

Characterizing properties of games

Key Ideas in Games

Defining Games

Characterizing properties of games

Simultaneous

Key Ideas in Games

Defining Games

Characterizing properties of games

Simultaneous

Sequential

Key Ideas in Games

Defining Games

Characterizing properties of games

Simultaneous

Sequential

What is difference between two?

Key Ideas in Games

Defining Games

Characterizing properties of games

Simultaneous

Sequential

What is difference between two?

Normal Form

Key Ideas in Games

Defining Games

Characterizing properties of games

Simultaneous

Sequential

What is difference between two?

Normal Form

Minimax Search

α-β pruning

Heuristic Search

Key Ideas in Games

Defining Games

Characterizing properties of games

Simultaneous

Sequential

What is difference between two?

Normal Form

Minimax Search

α-β pruning

Heuristic Search

Dominant Strategies
Best Responses
Pure vs. Mixed Strategies
Equilibria Concepts: DSE and Nash Eq

Outline

- Sequential-move games
  - Game trees, minimax, search approaches
- Speeding up sequential-move game search
  - Pruning, heuristics

Sequential-Move Games

More complex games with multiple moves

Sequential-Move Games

More complex games with multiple moves

- Instead of normal form, **extensive form**

Sequential-Move Games

More complex games with multiple moves

- Instead of normal form, **extensive form**
- Represent with a **tree**

Sequential-Move Games

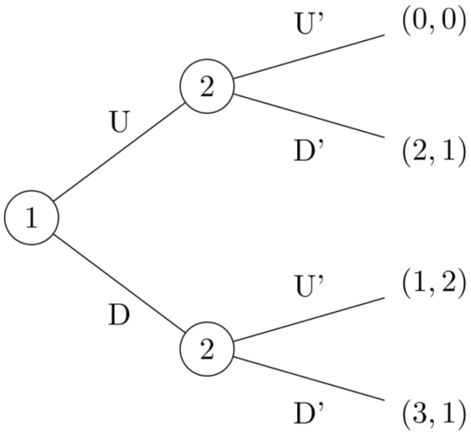
More complex games with multiple moves

- Instead of normal form, **extensive form**
- Represent with a **tree**
- **Rewards / pay-offs at leaves**

Sequential-Move Games

More complex games with multiple moves

- Instead of normal form, **extensive form**
- Represent with a **tree**
- **Rewards / pay-offs at leaves**

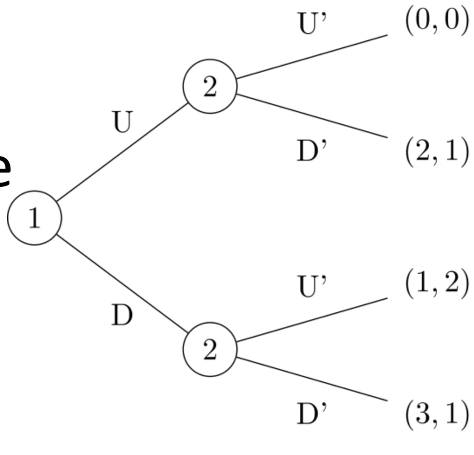


Wiki

Sequential-Move Games

More complex games with multiple moves

- Instead of normal form, **extensive form**
- Represent with a **tree**
- **Rewards / pay-offs at leaves**
- Find strategies: perform search over the tree



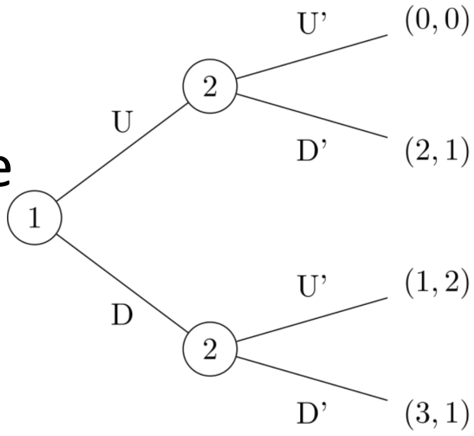
Wiki

Sequential-Move Games

More complex games with multiple moves

- Instead of normal form, **extensive form**
- Represent with a **tree**
- **Rewards / pay-offs at leaves**
- Find strategies: perform search over the tree

- Nash equilibrium still well-defined

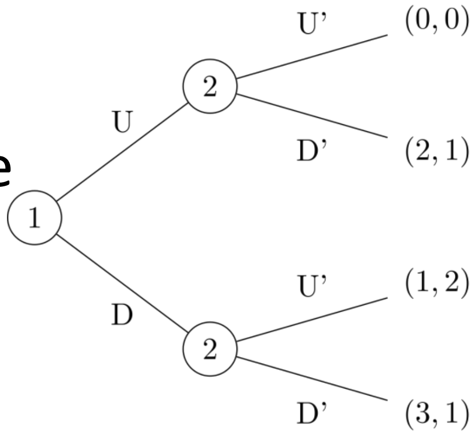


Wiki

Sequential-Move Games

More complex games with multiple moves

- Instead of normal form, **extensive form**
- Represent with a **tree**
- **Rewards / pay-offs at leaves**
- Find strategies: perform search over the tree



- Nash equilibrium still well-defined
  – Backward induction

Wiki

II-Nim: Example Sequential-Move Game

II-Nim: Example Sequential-Move Game

2 piles of sticks, each with 2 sticks.

II-Nim: Example Sequential-Move Game

2 piles of sticks, each with 2 sticks.

- Each player takes one or more sticks from pile

II-Nim: Example Sequential-Move Game

2 piles of sticks, each with 2 sticks.

- Each player takes one or more sticks from pile
- Take last stick: lose

II-Nim: Example Sequential-Move Game

2 piles of sticks, each with 2 sticks.

- Each player takes one or more sticks from pile
- Take last stick: lose

(ii, ii)

II-Nim: Example Sequential-Move Game

2 piles of sticks, each with 2 sticks.

- Each player takes one or more sticks from pile
- Take last stick: lose

(ii, ii)

- Two players: Max and Min

II-Nim: Example Sequential-Move Game

2 piles of sticks, each with 2 sticks.

- Each player takes one or more sticks from pile
- Take last stick: lose

(ii, ii)

- Two players: Max and Min
- If Max wins, its score is **+1**; otherwise **-1**

II-Nim: Example Sequential-Move Game

2 piles of sticks, each with 2 sticks.

- Each player takes one or more sticks from pile
- Take last stick: lose

(ii, ii)

- Two players: Max and Min
- If Max wins, its score is **+1**; otherwise **-1**
- Min's score is -1 * Max's (two-player zero-sum)

II-Nim: Example Sequential-Move Game

2 piles of sticks, each with 2 sticks.

- Each player takes one or more sticks from pile
- Take last stick: lose

(ii, ii)

- Two players: Max and Min
- If Max wins, its score is **+1**; otherwise **-1**
- Min's score is -1 * Max's (two-player zero-sum)
- Use Max's as the score of the game

Game Trajectory

(ii, ii)

Game Trajectory

(ii, ii)

Max takes one stick from one pile

(i, ii)

Game Trajectory

(ii, ii)

Max takes one stick from one pile

(i, ii)

Min takes two sticks from the other pile

(i,-)

Game Trajectory

(ii, ii)

Max takes one stick from one pile

(i, ii)

Min takes two sticks from the other pile

(i,-)

Max takes the last stick

(-,-)

Max gets score **-1**

Game tree for II-Nim

Two players:
Max and Min

(ii ii) Max

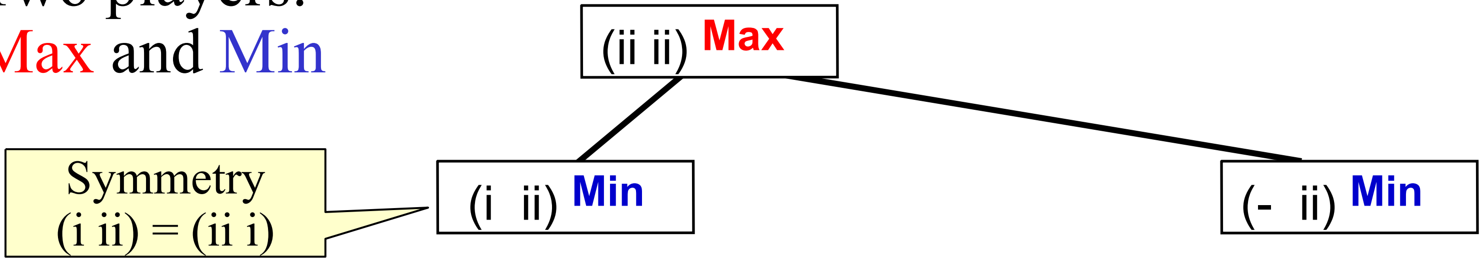
who is to move
at this state

Convention: score is w.r.t. the first player Max. Min's score = – Max

Max wants the largest score
Min wants the smallest score

Game tree for II-Nim

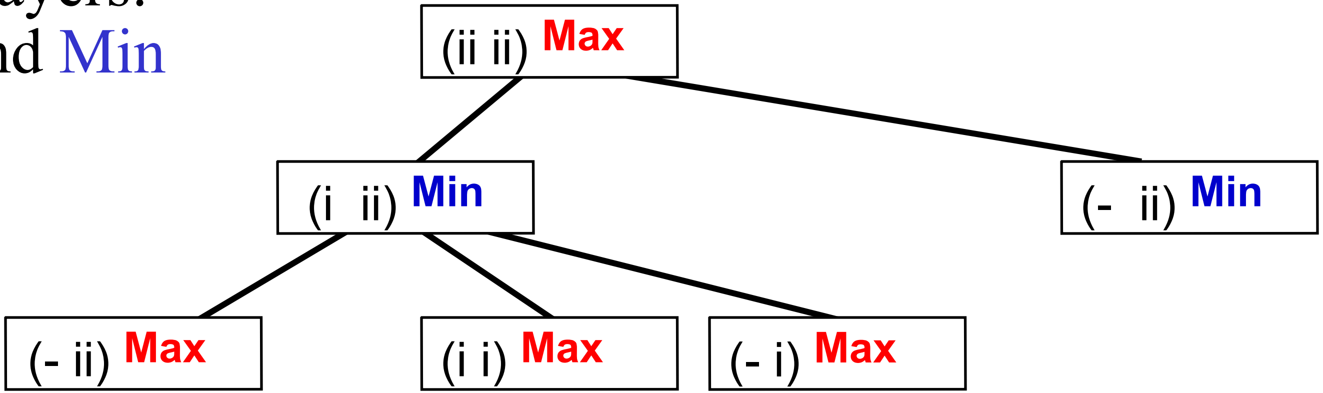
Two players:
Max and Min



Max wants the largest score
Min wants the smallest score

Game tree for II-Nim

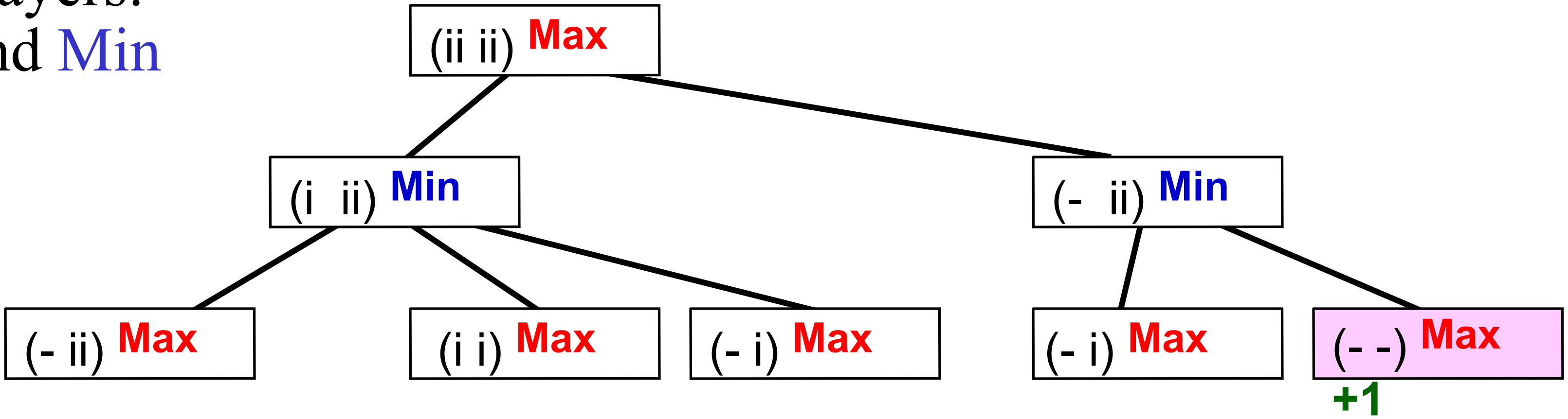
Two players:
Max and Min



Max wants the largest score
Min wants the smallest score

Game tree for II-Nim

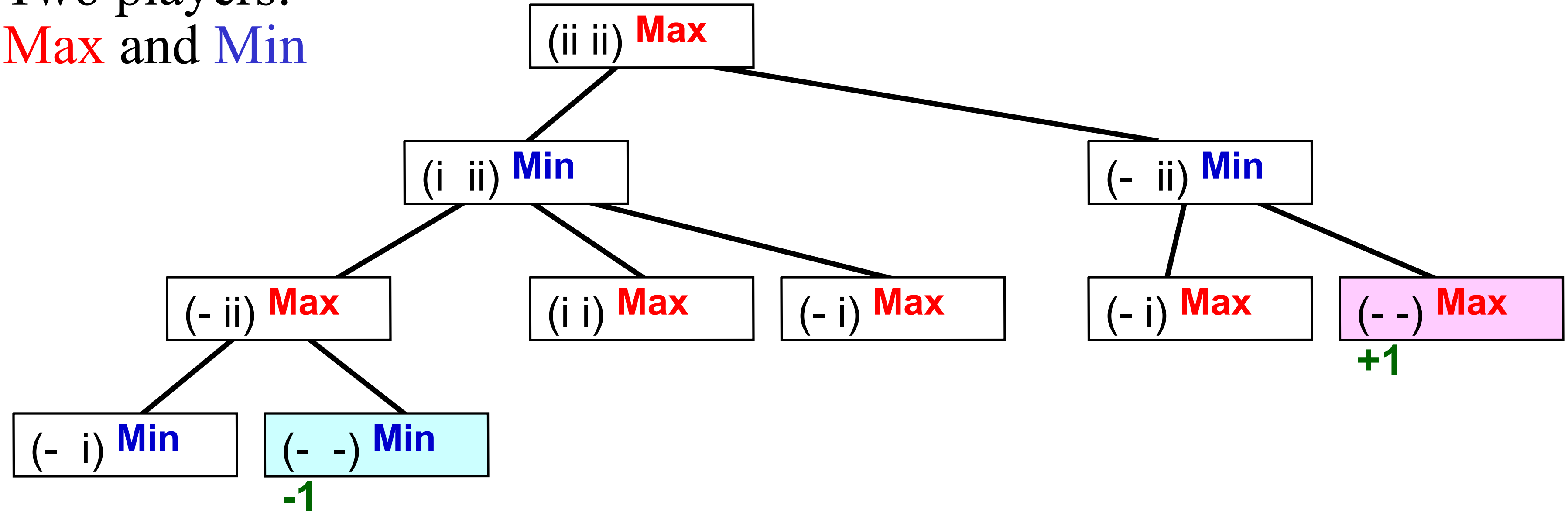
Two players:
Max and Min



Max wants the largest score
Min wants the smallest score

Game tree for II-Nim

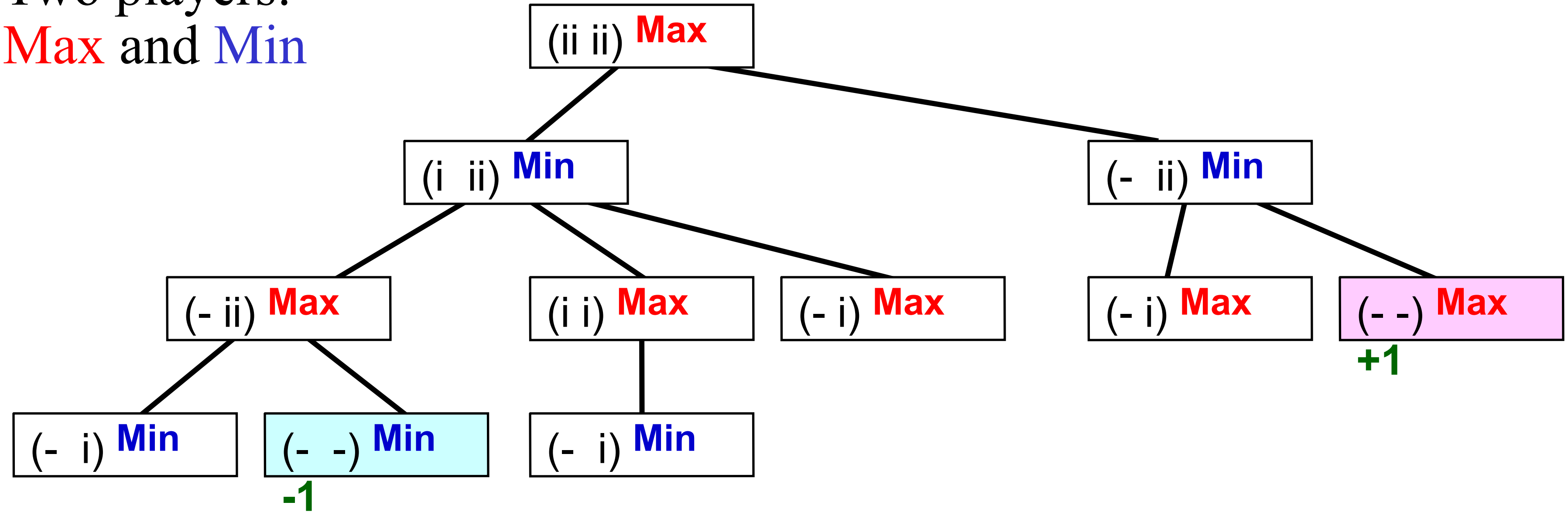
Two players:
Max and Min



Max wants the largest score
Min wants the smallest score

Game tree for II-Nim

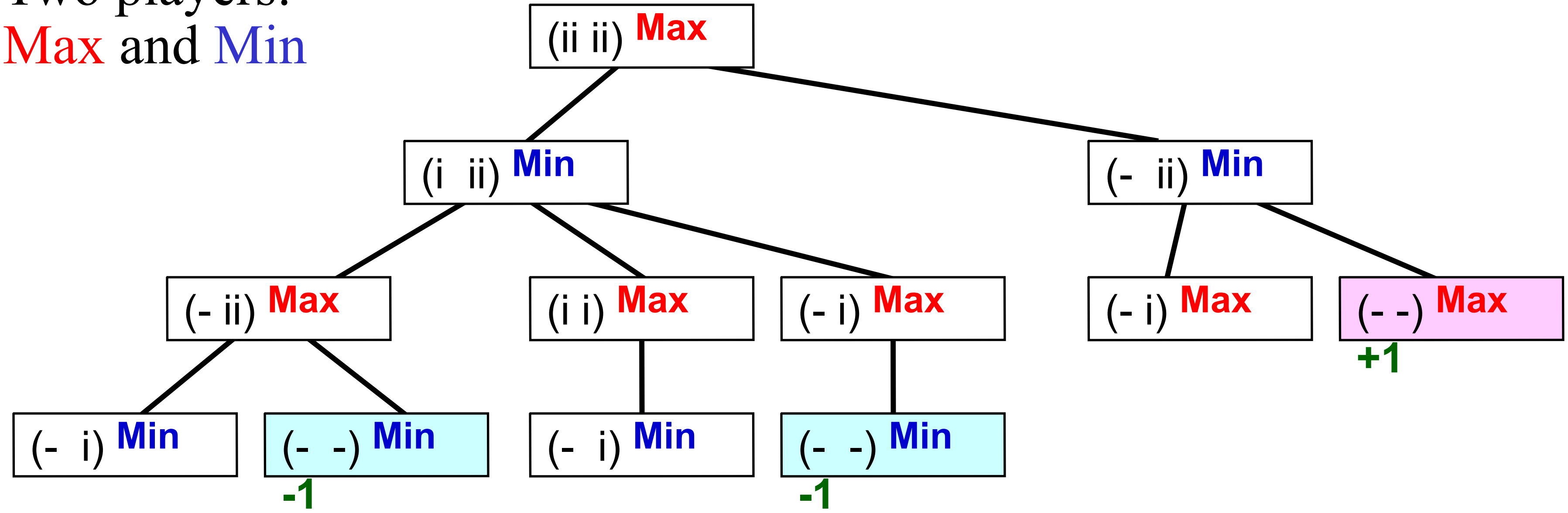
Two players:
Max and Min



Max wants the largest score
Min wants the smallest score

Game tree for II-Nim

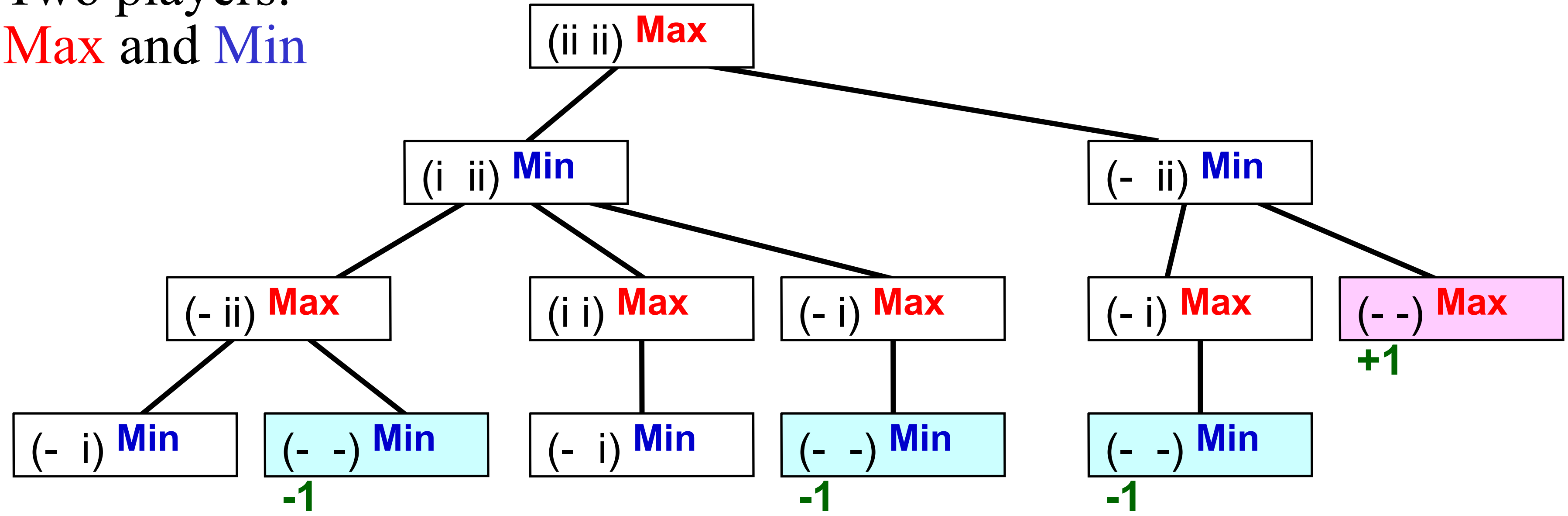
Two players:
Max and Min



Max wants the largest score
Min wants the smallest score

Game tree for II-Nim

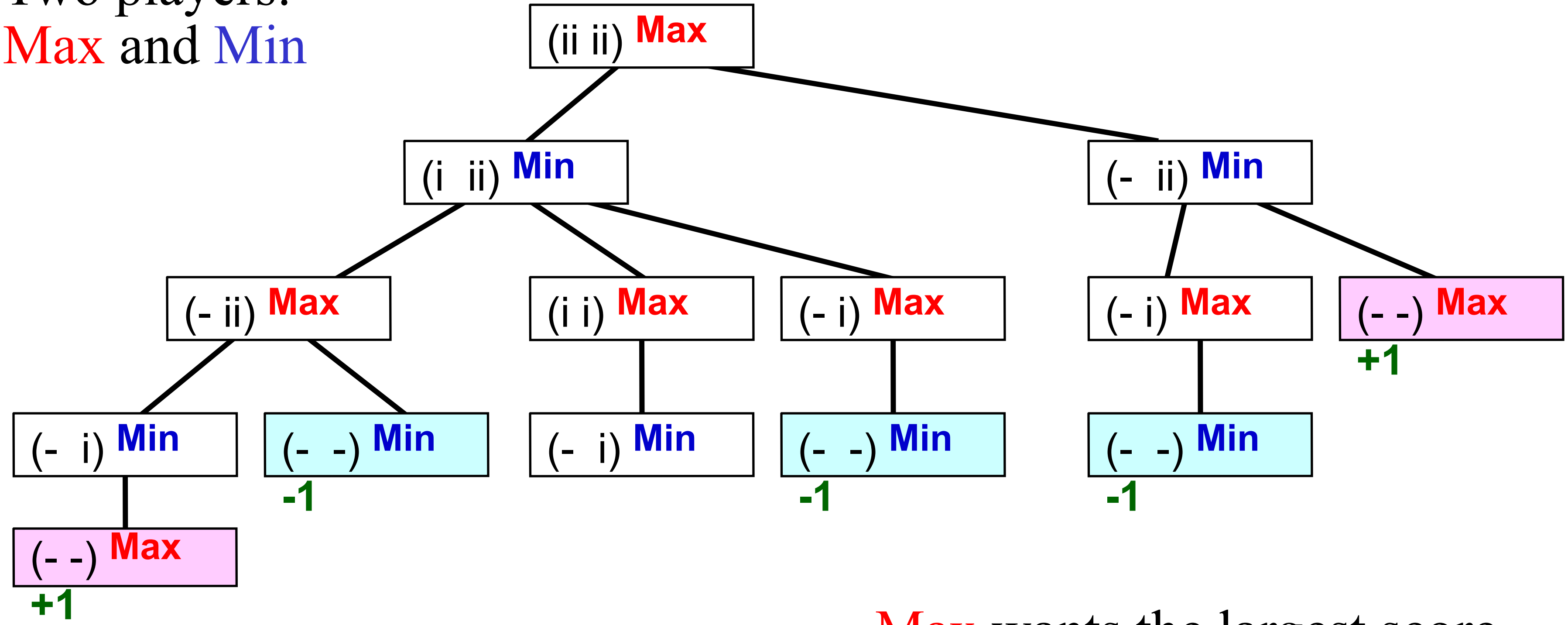
Two players:
Max and Min



Max wants the largest score
Min wants the smallest score

Game tree for II-Nim

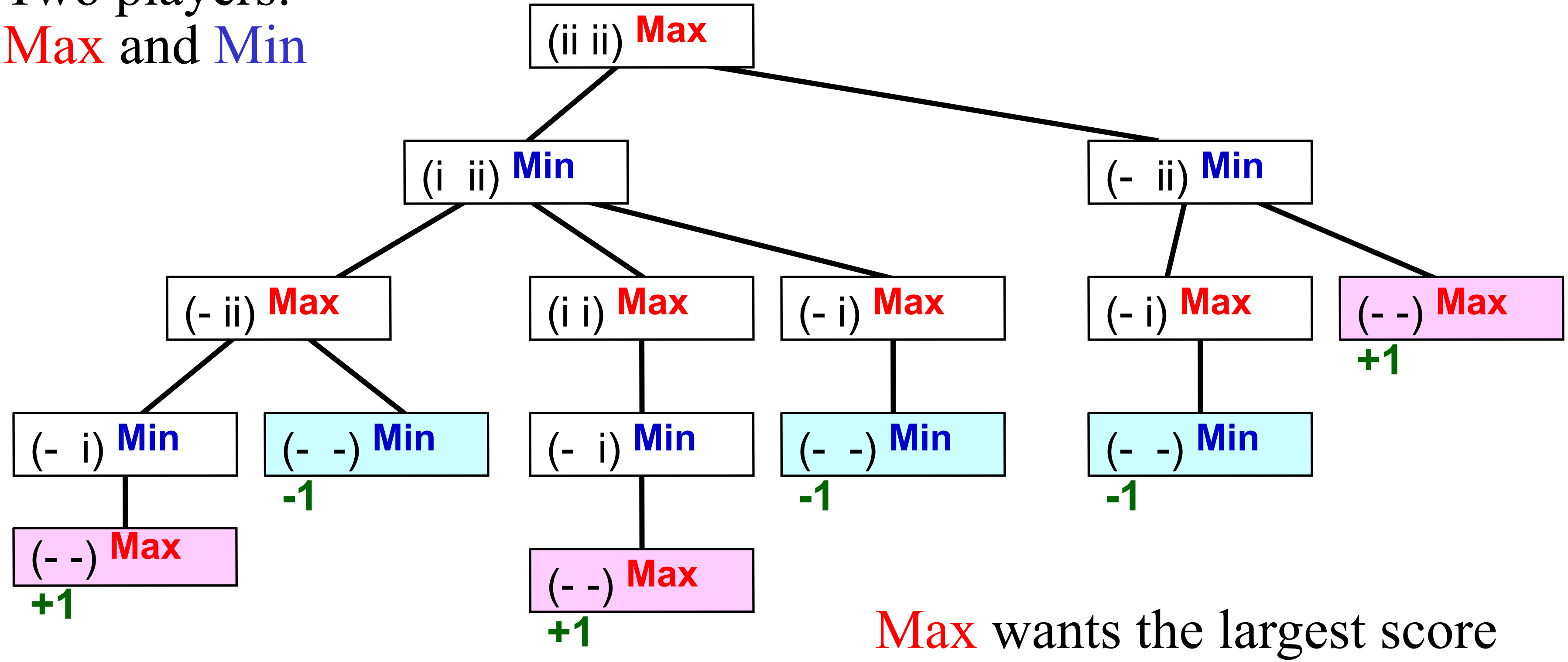
Two players:
Max and Min



Max wants the largest score
Min wants the smallest score

Game tree for II-Nim

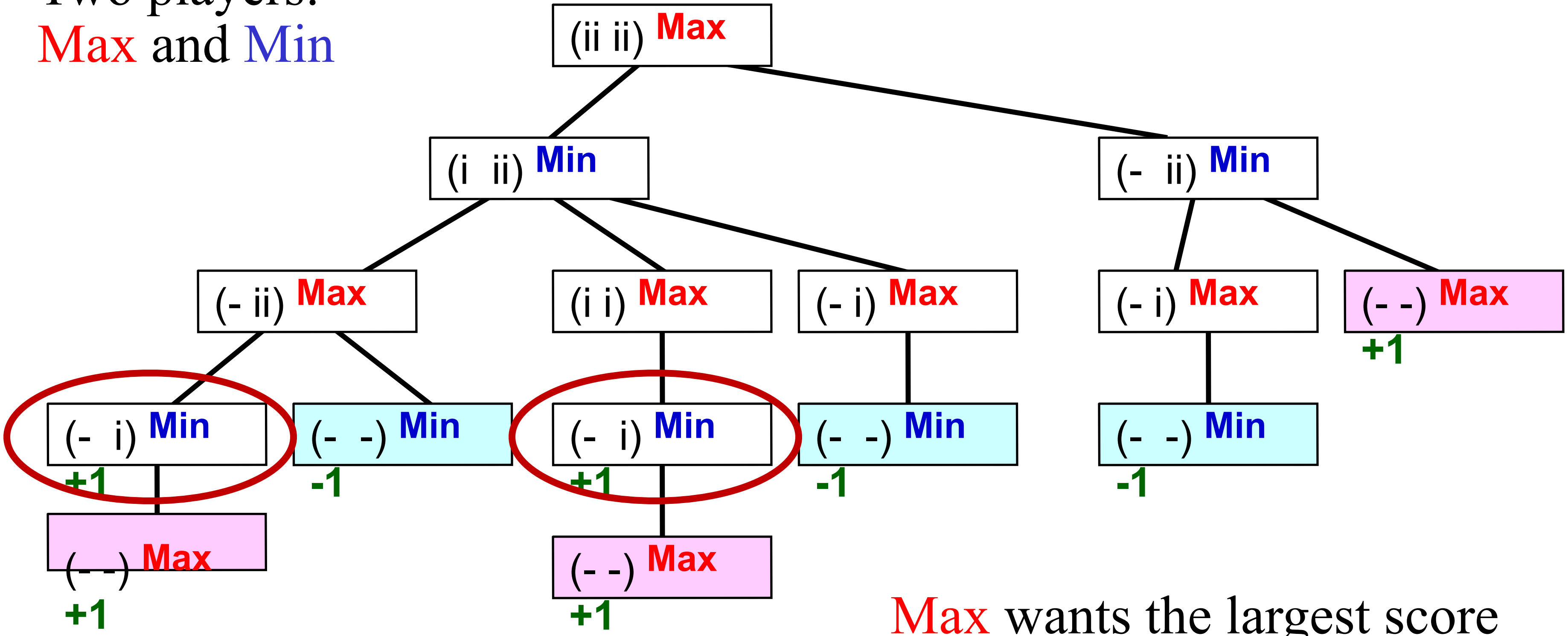
Two players:
Max and Min



Max wants the largest score
Min wants the smallest score

Game tree for II-Nim

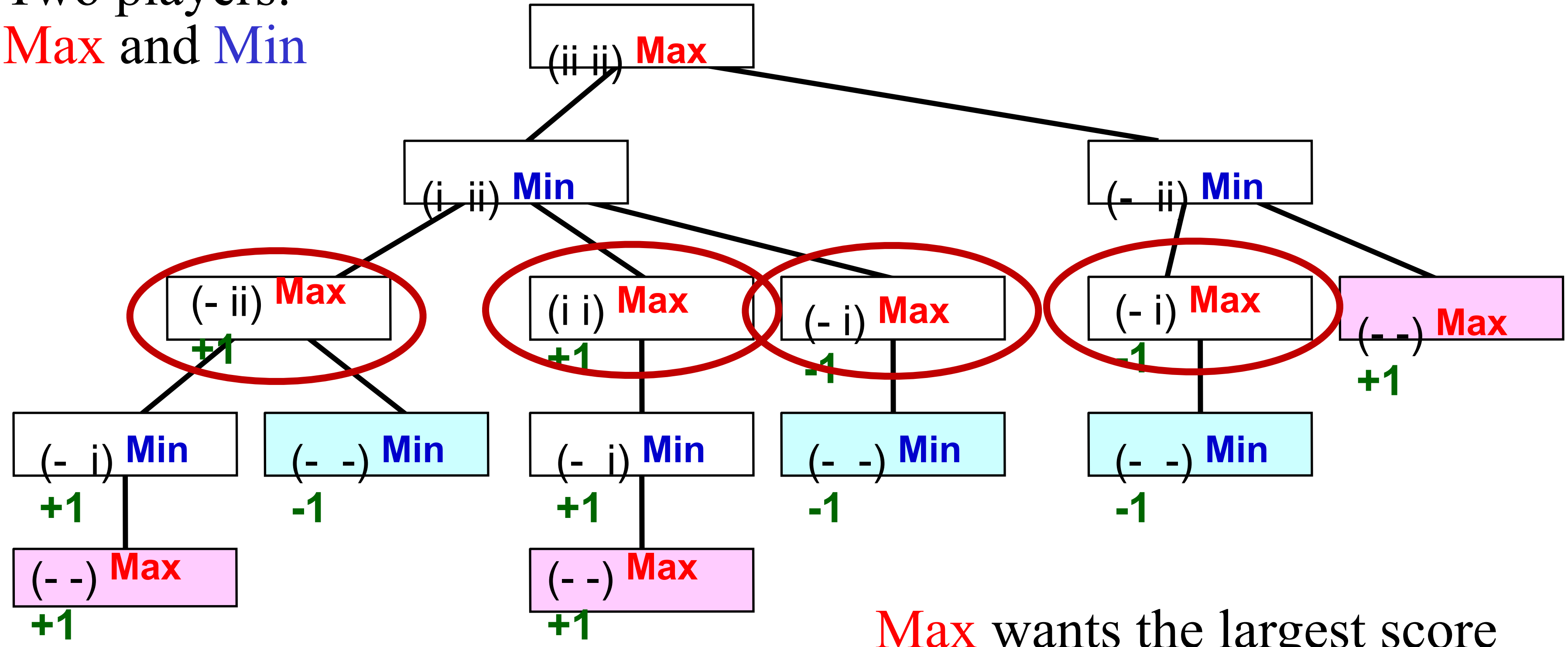
Two players:
Max and Min



Max wants the largest score
Min wants the smallest score

Game tree for II-Nim

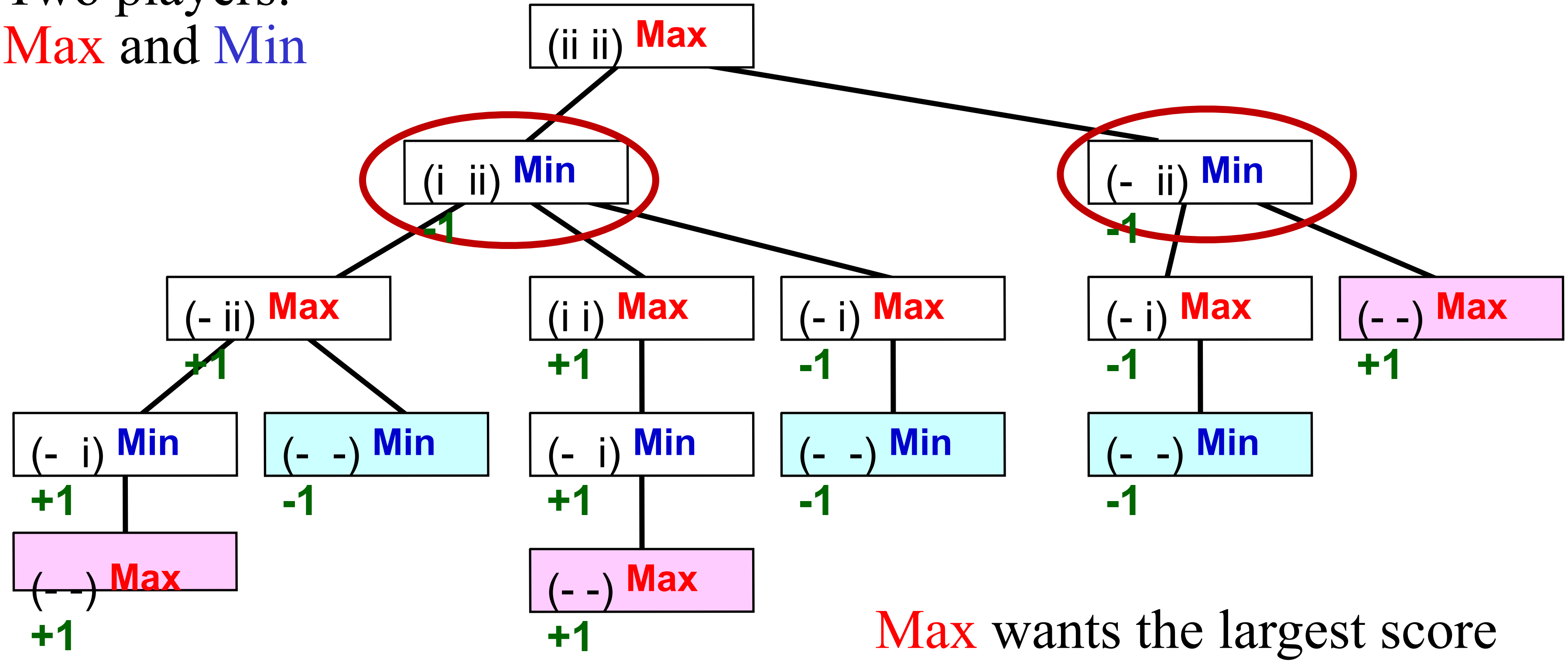
Two players:
Max and Min



Max wants the largest score
Min wants the smallest score

Game tree for II-Nim

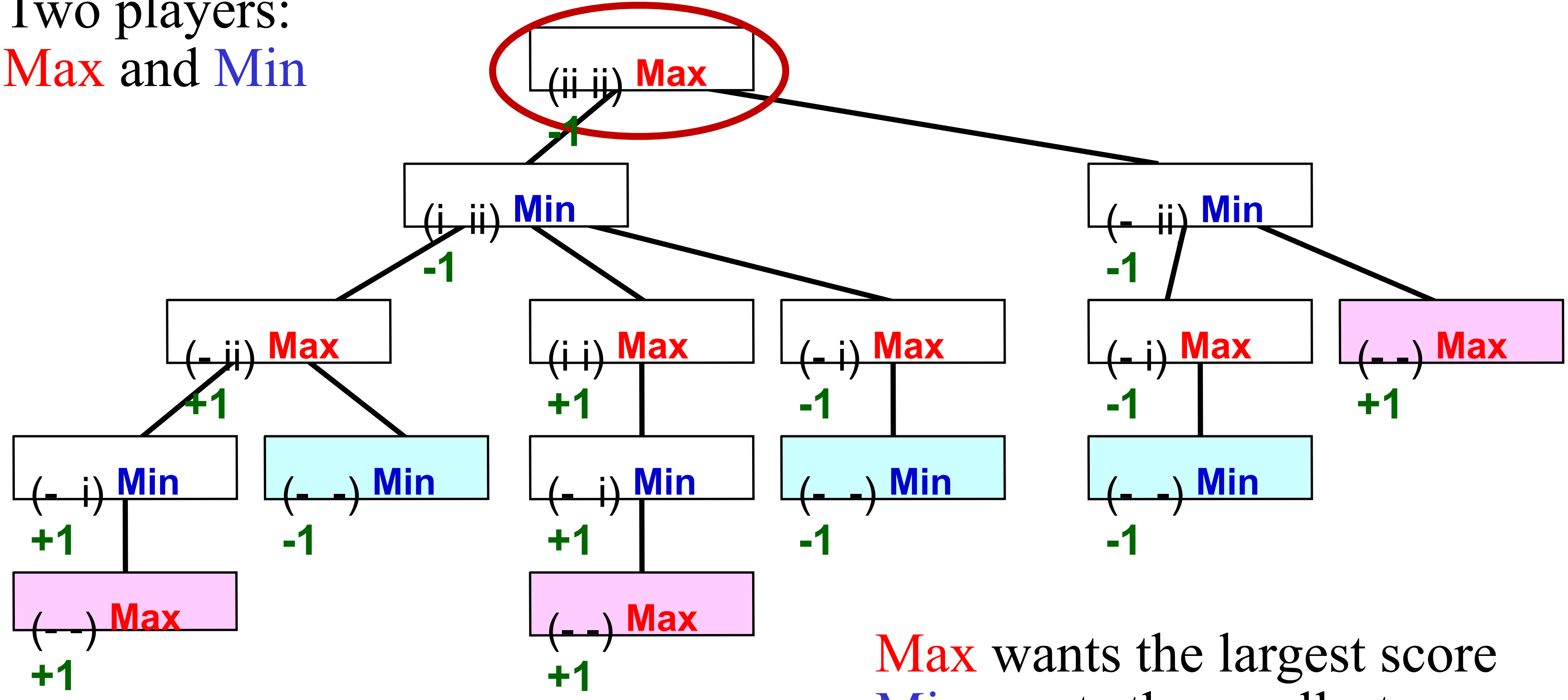
Two players:
Max and Min



Max wants the largest score
Min wants the smallest score

Game tree for II-Nim

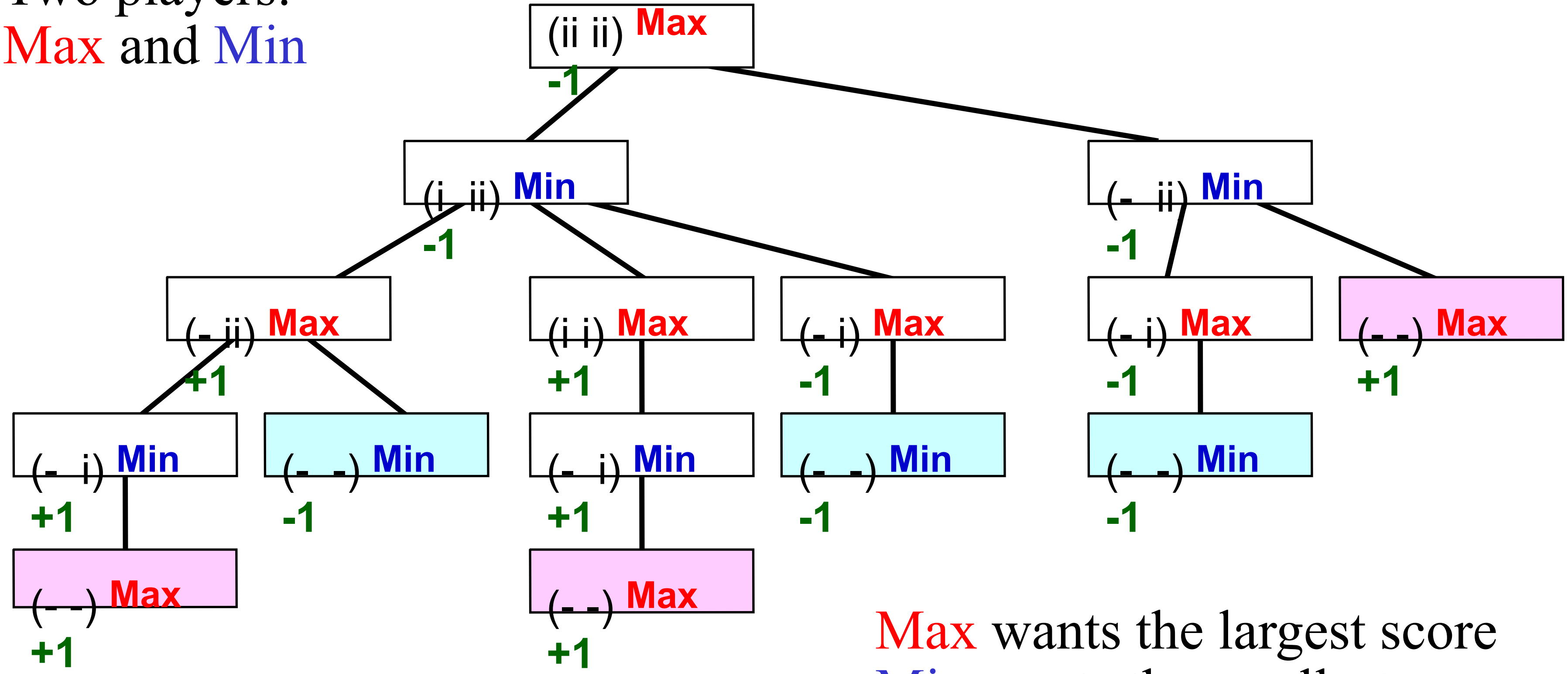
Two players:
Max and Min



Max wants the largest score
Min wants the smallest score

Game tree for II-Nim

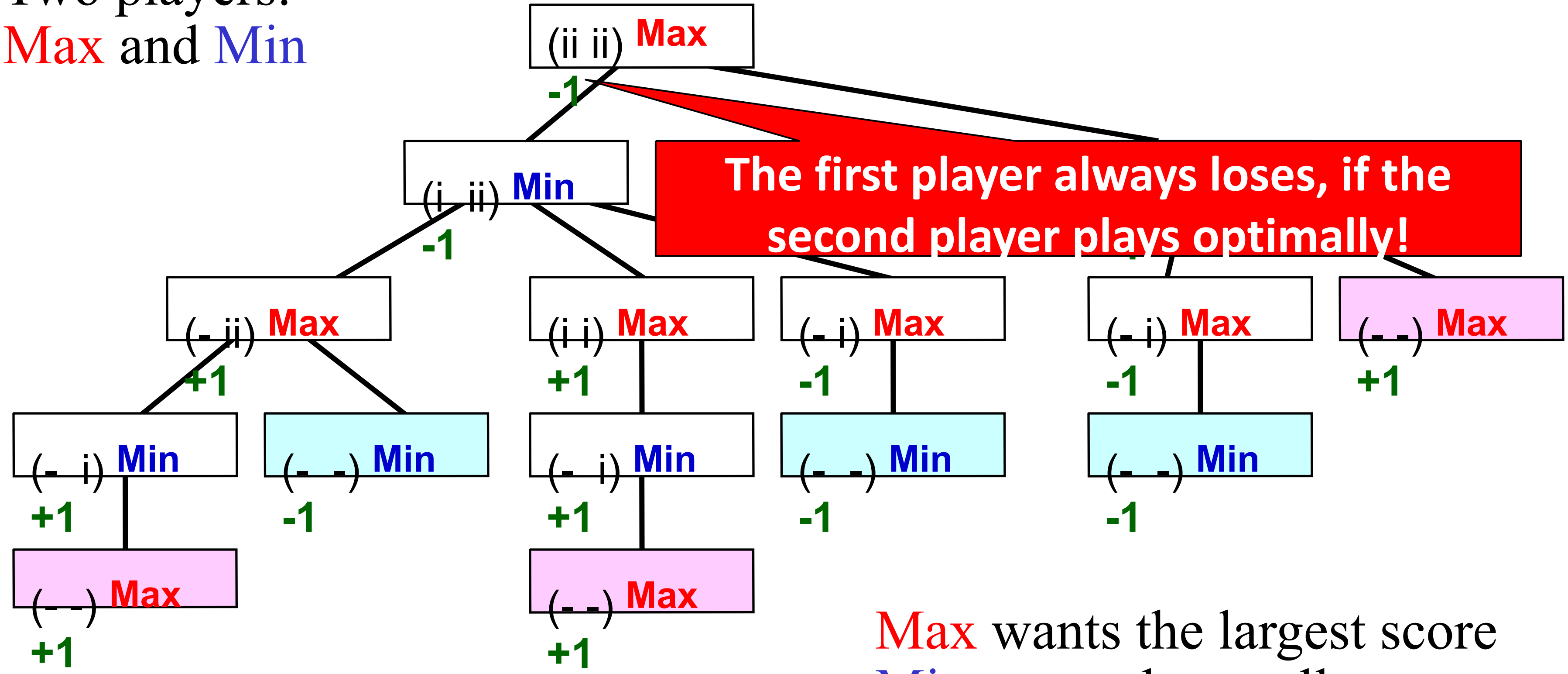
Two players:
Max and Min



Max wants the largest score
Min wants the smallest score

Game tree for II-Nim

Two players:
Max and Min



Max wants the largest score
Min wants the smallest score

Our Approach So Far

We find the minimax value/strategy bottom up

Our Approach So Far

We find the minimax value/strategy bottom up

- Minimax value: score of terminal node when both players play optimally

Our Approach So Far

We find the minimax value/strategy bottom up

- Minimax value: score of terminal node when both players play optimally
  - **Max's** turn, take max of children

Our Approach So Far

We find the minimax value/strategy bottom up

- Minimax value: score of terminal node when both players play optimally
  - **Max's** turn, take max of children
  - **Min's** turn, take min of children

Our Approach So Far

We find the minimax value/strategy bottom up

- Minimax value: score of terminal node when both players play optimally
  - **Max's** turn, take max of children
  - **Min's** turn, take min of children

- Can implement this as depth-first search: **minimax algorithm**

Minimax Algorithm

```
function Max-Value(s)
inputs:
      s: current state in game, Max about to play
output: best-score (for Max) available from s

      if ( s is a terminal state )
      then return ( terminal value of s )
      else
      α := – infinity
      for each s’ in Succ(s)
          α := max( α , Min-value(s’))
      return α
```

```
function Min-Value(s)
output: best-score (for Min) available from s

      if ( s is a terminal state )
      then return ( terminal value of s)
      else
      β := infinity
      for each s’ in Succs(s)
          β := min( β , Max-value(s’))
      return β
```

Minimax Algorithm

```
function Max-Value(s)
inputs:
    s: current state in game, Max about to play
output: best-score (for Max) available from s

    if ( s is a terminal state )
    then return ( terminal value of s )
    else
    α := – infinity
    for each s’ in Succ(s)
        α := max( α , Min-value(s’))
    return α
```

```
function Min-Value(s)
output: best-score (for Min) available from s

    if ( s is a terminal state )
    then return ( terminal value of s)
    else
    β := infinity
    for each s’ in Succs(s)
        β := min( β , Max-value(s’))
    return β
```

Time complexity?

Minimax Algorithm

```
function Max-Value(s)
inputs:
      s: current state in game, Max about to play
output: best-score (for Max) available from s

      if ( s is a terminal state )
      then return ( terminal value of s )
      else
      α := – infinity
      for each s' in Succ(s)
          α := max( α , Min-value(s'))
      return α
```

```
function Min-Value(s)
output: best-score (for Min) available from s

      if ( s is a terminal state )
      then return ( terminal value of s)
      else
      β := infinity
      for each s' in Succs(s)
          β := min( β , Max-value(s'))
      return β
```

Time complexity?

- $O(b^m)$

Minimax Algorithm

```
function Max-Value(s)
inputs:
      s: current state in game, Max about to play
output: best-score (for Max) available from s

      if ( s is a terminal state )
      then return ( terminal value of s )
      else
      α := – infinity
      for each s’ in Succ(s)
          α := max( α , Min-value(s’))
      return α
```

```
function Min-Value(s)
output: best-score (for Min) available from s

      if ( s is a terminal state )
      then return ( terminal value of s)
      else
      β := infinity
      for each s’ in Succs(s)
          β := min( β , Max-value(s’))
      return β
```

Time complexity?

- $O(b^m)$

Space complexity?

Minimax Algorithm

```
function Max-Value(s)
inputs:
      s: current state in game, Max about to play
output: best-score (for Max) available from s

      if ( s is a terminal state )
      then return ( terminal value of s )
      else
      α := – infinity
      for each s’ in Succ(s)
          α := max( α , Min-value(s’))
      return α
```

```
function Min-Value(s)
output: best-score (for Min) available from s

      if ( s is a terminal state )
      then return ( terminal value of s)
      else
      β := infinity
      for each s’ in Succs(s)
          β := min( β , Max-value(s’))
      return β
```

Time complexity?

- $O(b^m)$

Space complexity?

- $O(bm)$

Minimax Algorithm

```
function Max-Value(s)
inputs:
    s: current state in game, Max about to play
output: best-score (for Max) available from s

    if ( s is a terminal state )
    then return ( terminal value of s )
    else
    α := – infinity
    for each s’ in Succ(s)
        α := max( α , Min-value(s’))
    return α
```

```
function Min-Value(s)
output: best-score (for Min) available from s

    if ( s is a terminal state )
    then return ( terminal value of s)
    else
    β := infinity
    for each s’ in Succs(s)
        β := min( β , Max-value(s’))
    return β
```

Time complexity?

- $O(b^m)$

Space complexity?

- $O(bm)$

Break & Quiz

Q 2.1: We are playing a game where Player A goes first and has 4 moves. Player B goes next and has 3 moves. Player A goes next and has 2 moves. Player B then has one move.

How many nodes are there in the minimax tree, including termination nodes (leaves)?

- A. 23
- B. 65
- C. 41
- D. 2

Break & Quiz

Q 2.1: We are playing a game where Player A goes first and has 4 moves. Player B goes next and has 3 moves. Player A goes next and has 2 moves. Player B then has one move.

How many nodes are there in the minimax tree, including termination nodes (leaves)?

- A. 23
- **B. 65**
- C. 41
- D. 2

Break & Quiz

Q 2.1: We are playing a game where Player A goes first and has 4 moves. Player B goes next and has 3 moves. Player A goes next and has 2 moves. Player B then has one move.

How many nodes are there in the minimax tree, including termination nodes (leaves)?

- A. 23
- **B. 65** (1 + 4 + 4*3 + 4*3*2 + 4*3*2 = 65. Note the root and leaf nodes.)
- C. 41
- D. 2

Break & Quiz

Q 2.2: During minimax tree search, must we examine every node?

- A. Always
- B. Sometimes
- C. Never

Break & Quiz

Q 2.2: During minimax tree search, must we examine every node?

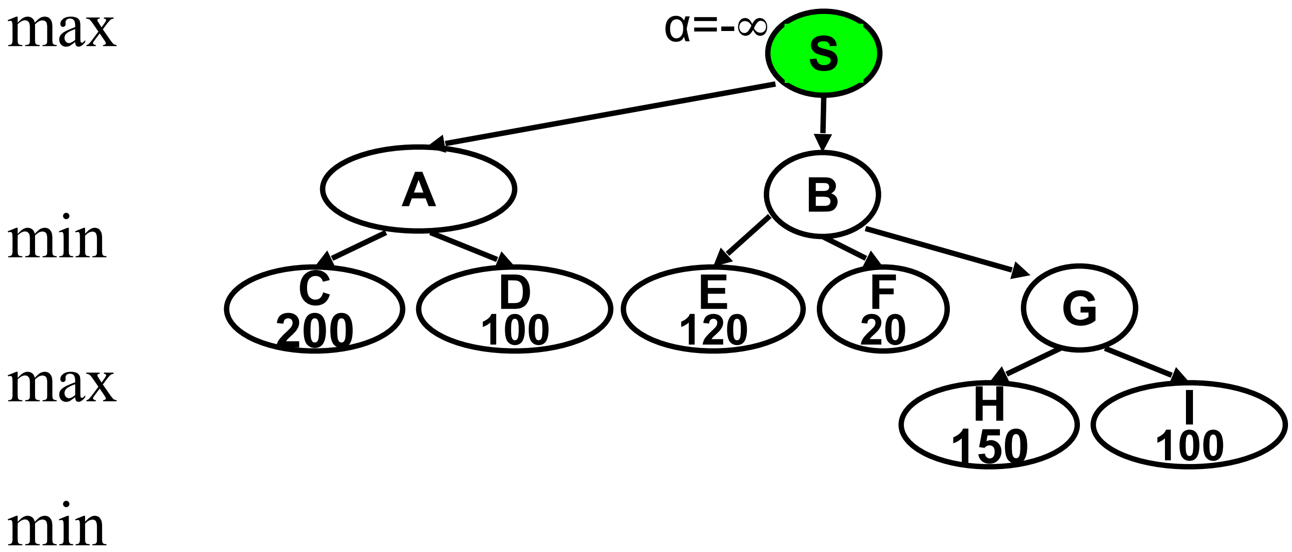
- A. Always
- **B. Sometimes**
- C. Never

Break & Quiz

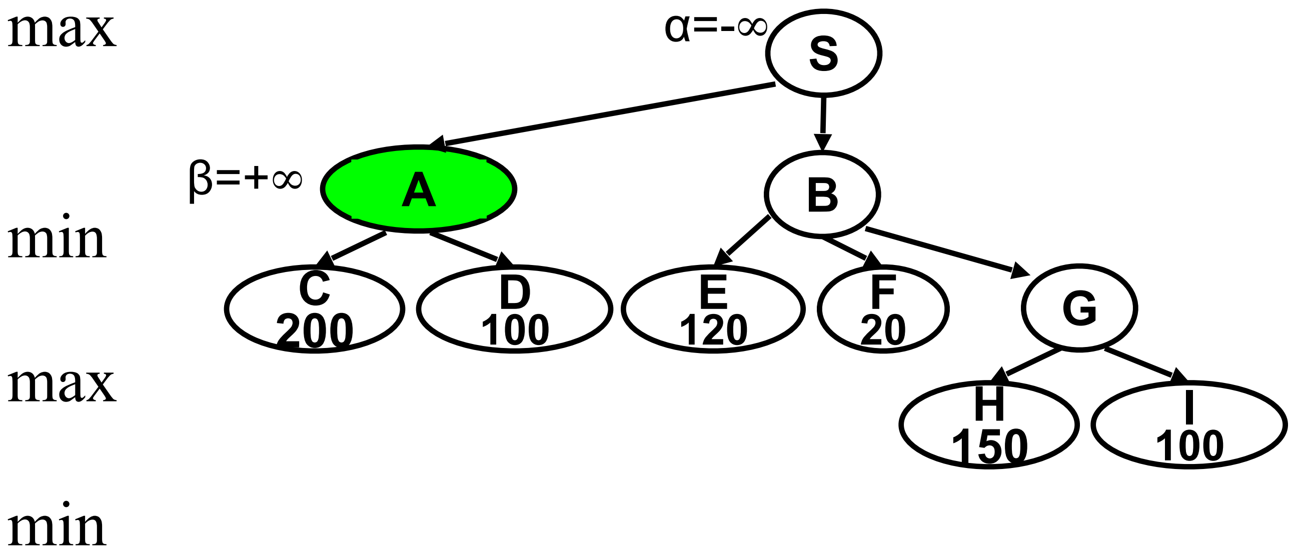
Q 2.2: During minimax tree search, must we examine every node?

- A. Always (No: consider layer *k*, where we take the max of all the mins of its children at layer *k+1*. If the current value of a min node at *k+1* already smaller than the current max, we don't need to continue the minimization.)
- **B. Sometimes**
- C. Never (No: the event above may simply not happen).

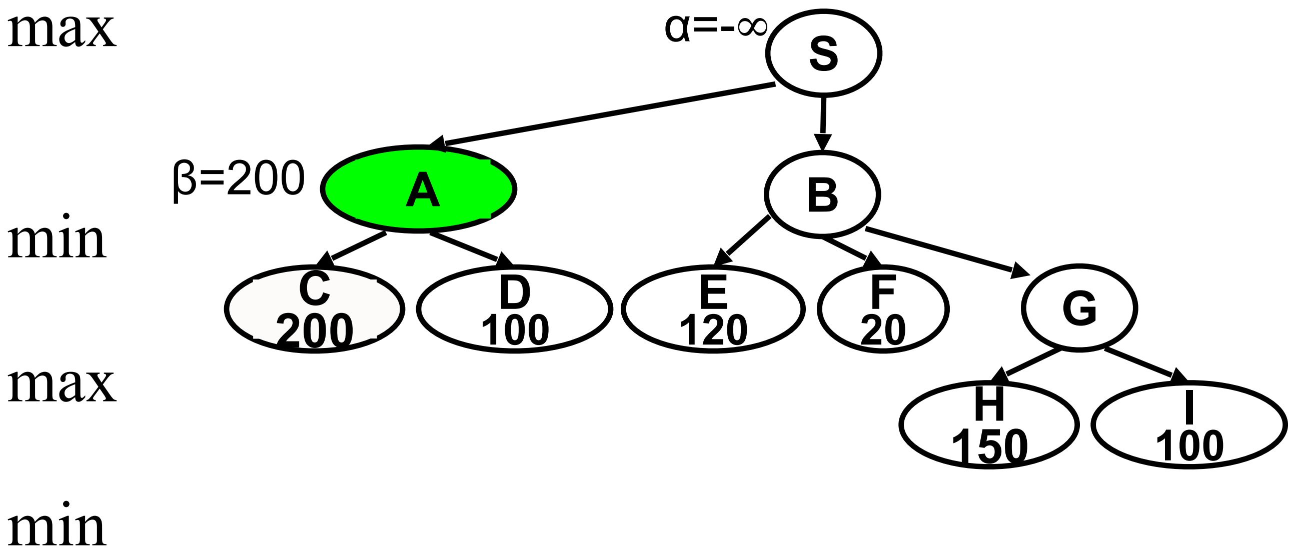
Minimax algorithm in execution



Minimax algorithm in execution

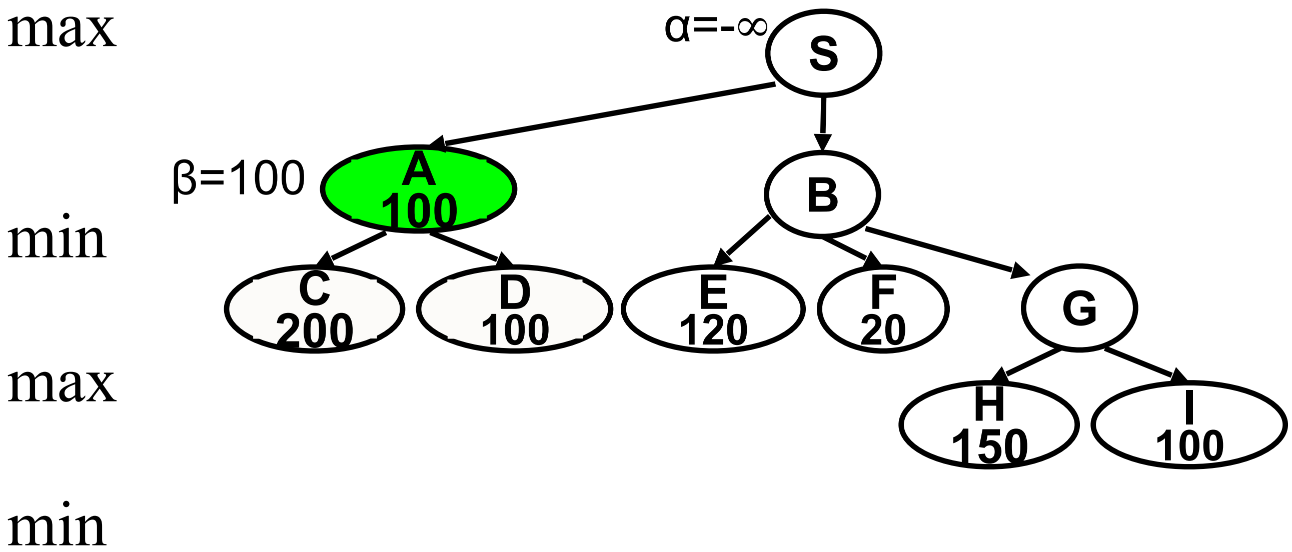


Minimax algorithm in execution

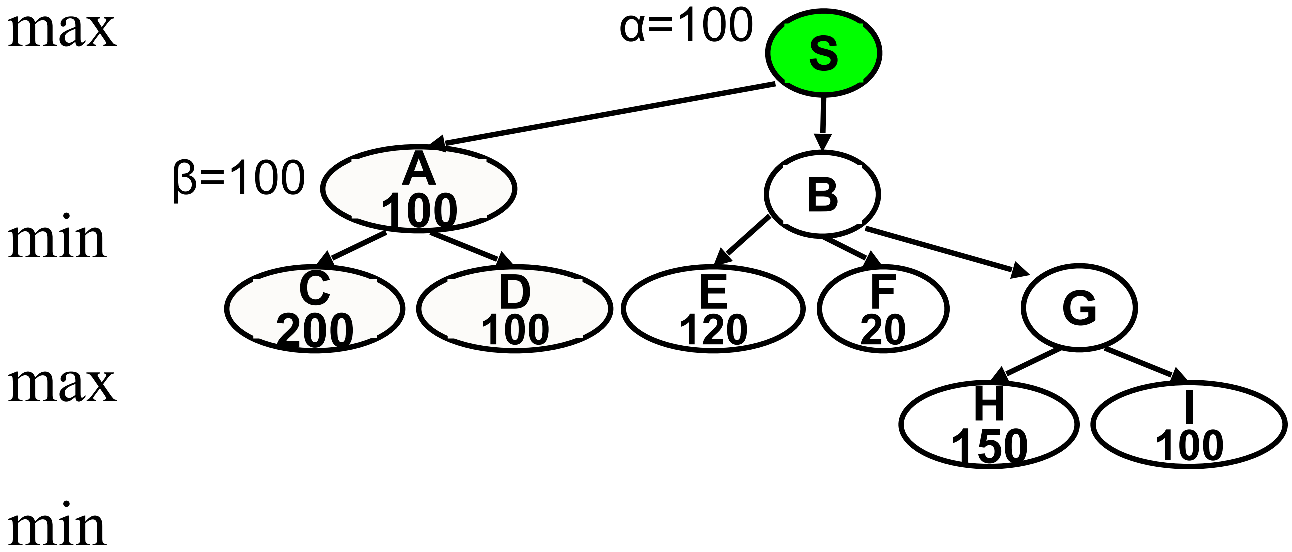


The execution on the
terminal nodes is omitted.

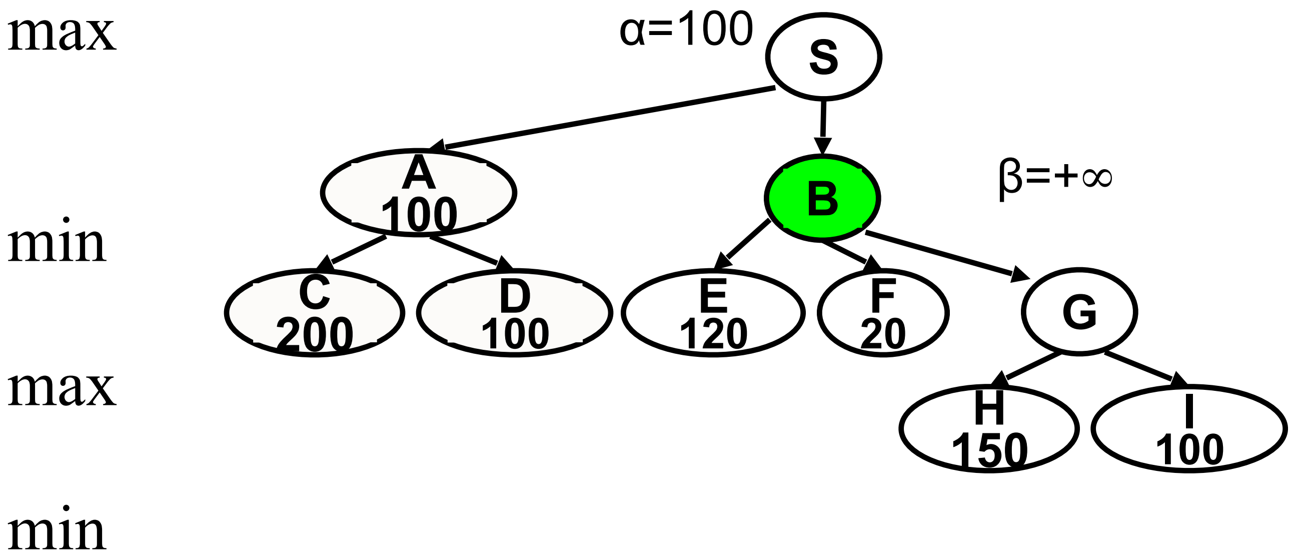
Minimax algorithm in execution



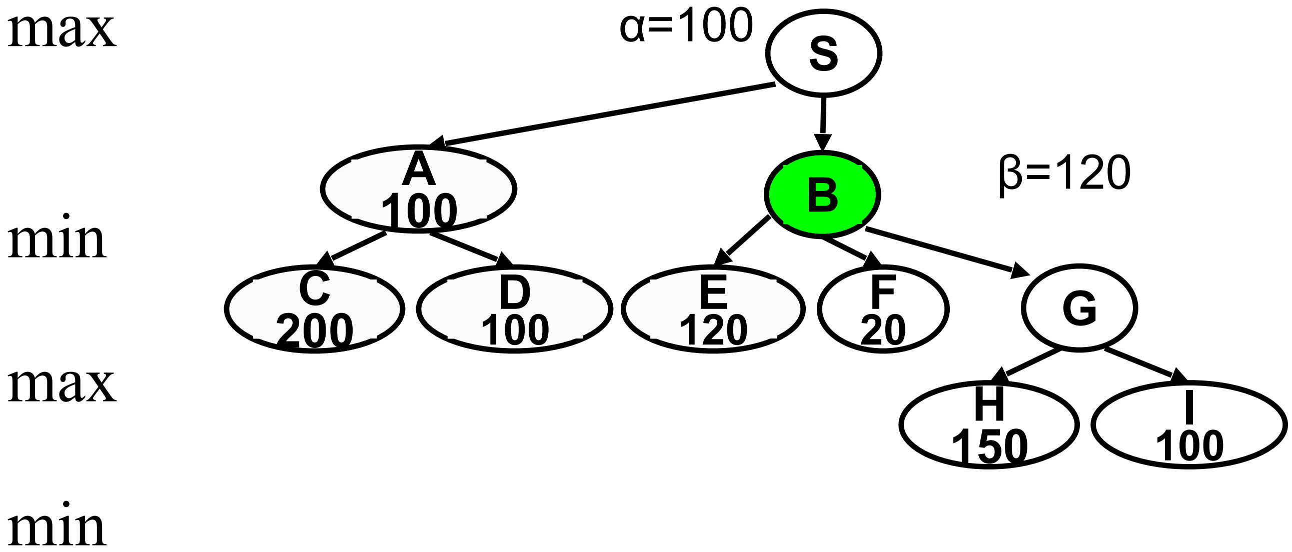
Minimax algorithm in execution



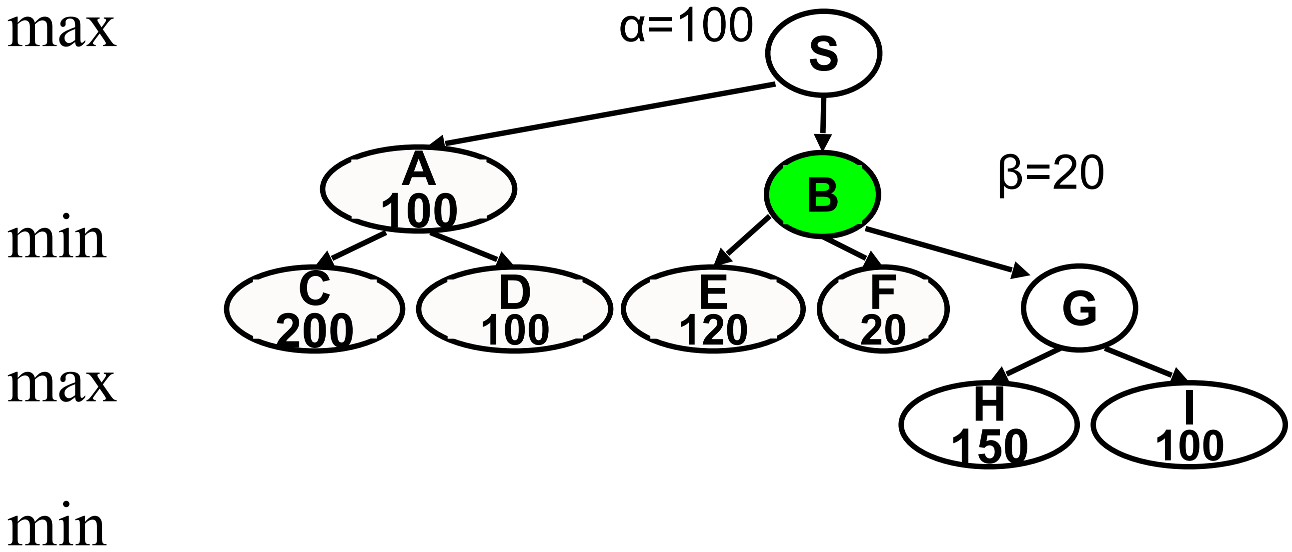
Minimax algorithm in execution



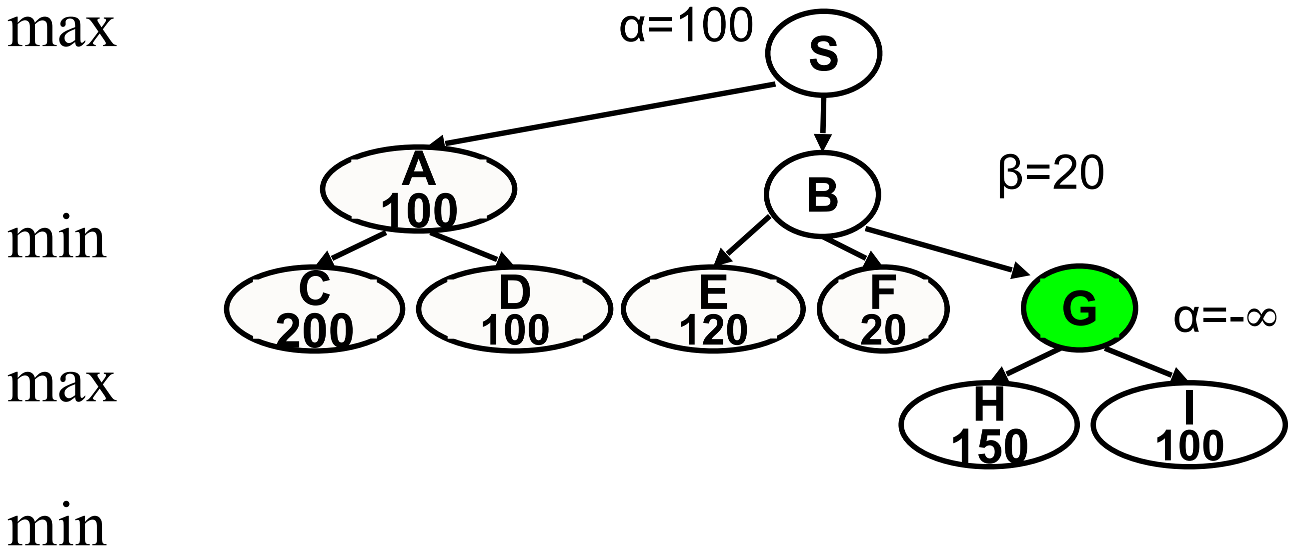
Minimax algorithm in execution



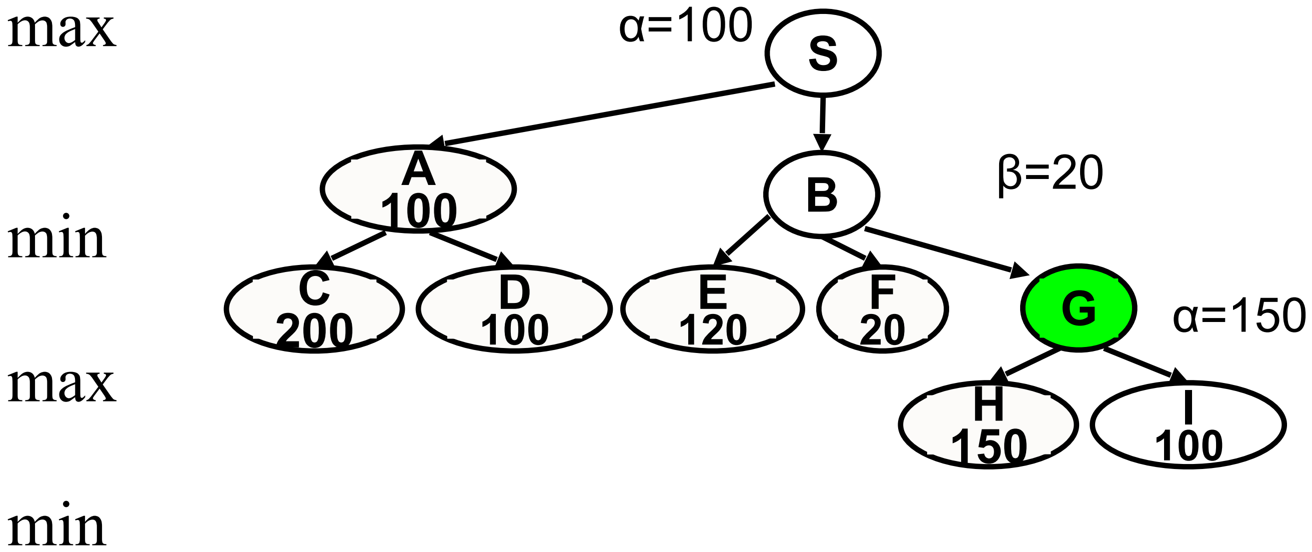
Minimax algorithm in execution



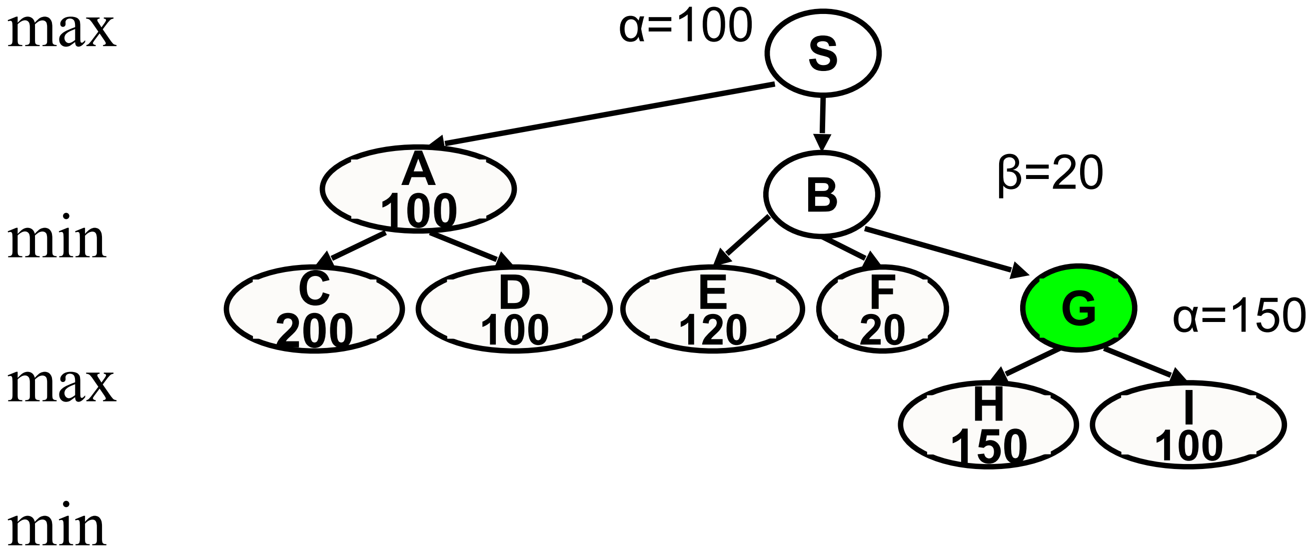
Minimax algorithm in execution



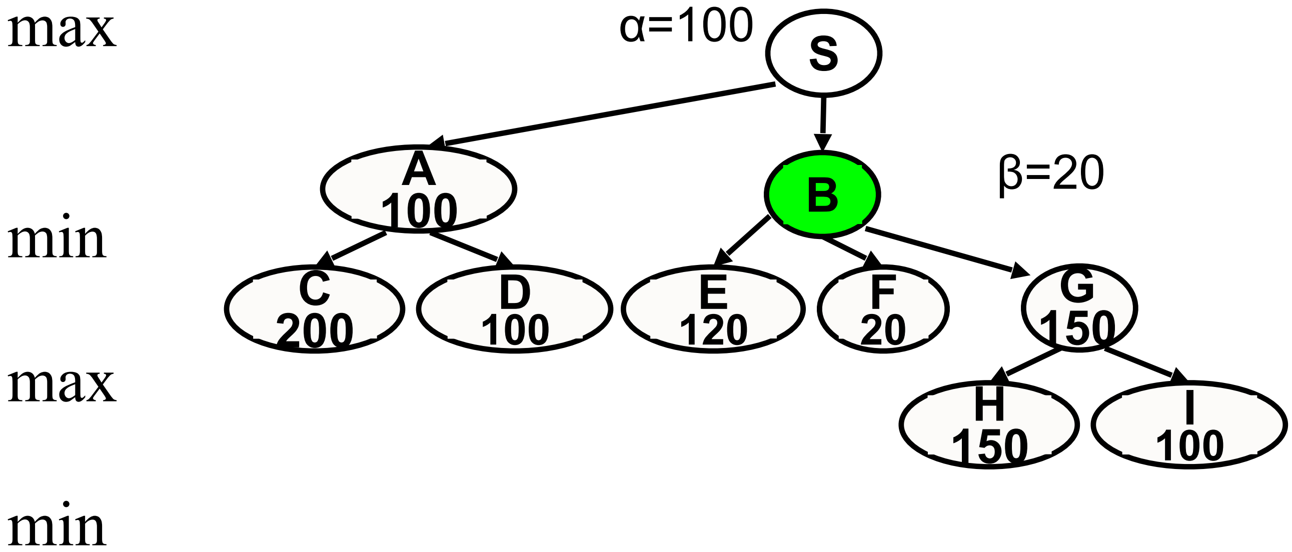
Minimax algorithm in execution



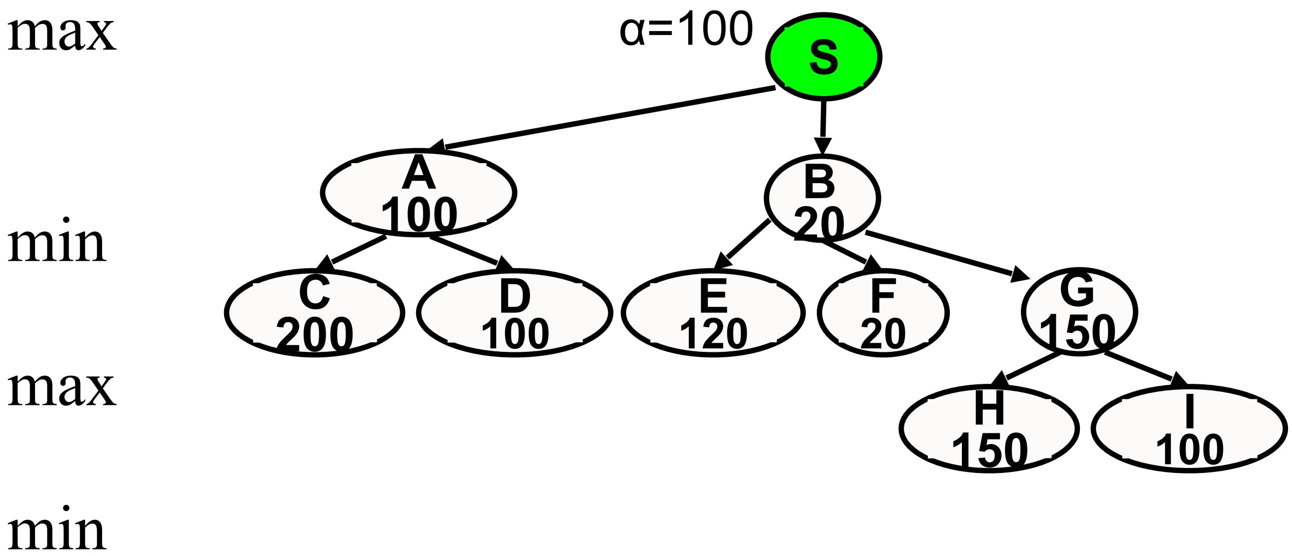
Minimax algorithm in execution



Minimax algorithm in execution



Minimax algorithm in execution



Can We Do Better?

One **downside**: we had to examine the entire tree

Can We Do Better?

One **downside**: we had to examine the entire tree

An idea to speed things up: **pruning**

Can We Do Better?

One **downside**: we had to examine the entire tree

An idea to speed things up: **pruning**

- Goal: want the same minimax value, but faster

Can We Do Better?

One **downside**: we had to examine the entire tree

An idea to speed things up: **pruning**

- Goal: want the same minimax value, but faster
- We can get rid of bad branches

Can We Do Better?

One **downside**: we had to examine the entire tree

An idea to speed things up: **pruning**

- Goal: want the same minimax value, but faster
- We can get rid of bad branches
- Same principle as quiz question 2.

Can We Do Better?

One **downside**: we had to examine the entire tree

An idea to speed things up: **pruning**

- Goal: want the same minimax value, but faster
- We can get rid of bad branches
- Same principle as quiz question 2.

Alpha-beta pruning

```
function Max-Value (s,α,β)
inputs:
    s: current state in game, Max about to play
    α: best score (highest) for Max along path to s
    β: best score (lowest) for Min along path to s
output: min(β , best-score (for Max) available from s)

    if ( s is a terminal state )
    then return ( terminal value of s )
    else for each s' in Succ(s)
      α := max( α , Min-value(s',α,β))
      if ( α ≥ β ) then return β   /* alpha pruning */
    return α
```

```
function Min-Value(s,α,β)
output: max(α , best-score (for Min) available from s )

    if ( s is a terminal state )
    then return ( terminal value of s)
    else for each s' in Succs(s)
      β := min( β , Max-value(s',α,β))
      if (α ≥ β ) then return α   /* beta pruning */
    return β
```

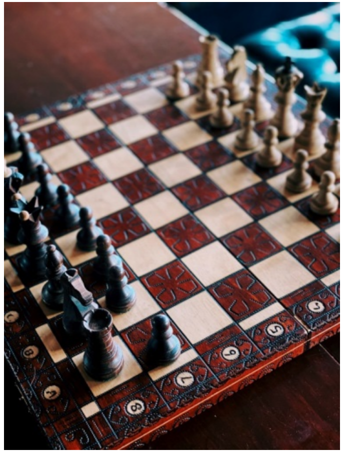
Starting from the root:
Max-Value(root, $-\infty$, $+\infty$)

Alpha-Beta Pruning

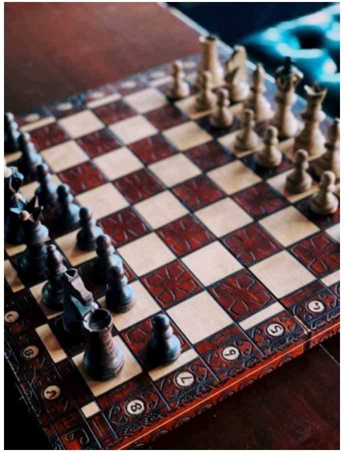
How effective is **alpha-beta pruning**?

Alpha-Beta Pruning

How effective is **alpha-beta pruning**?



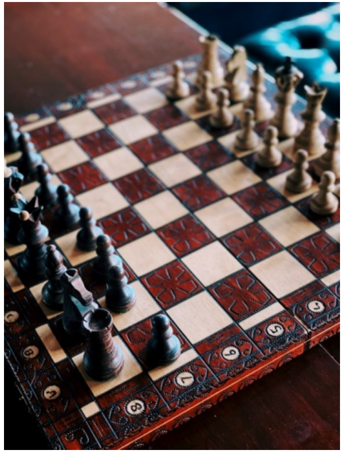
Alpha-Beta Pruning



How effective is **alpha-beta pruning**?

- Depends on the order of successors!

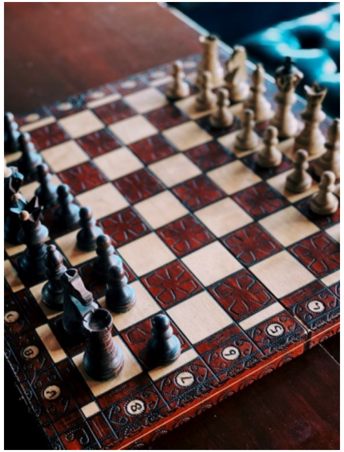
Alpha-Beta Pruning



How effective is **alpha-beta pruning**?

- Depends on the order of successors!
  - Best case, the # of nodes to search is $O(b^{m/2})$

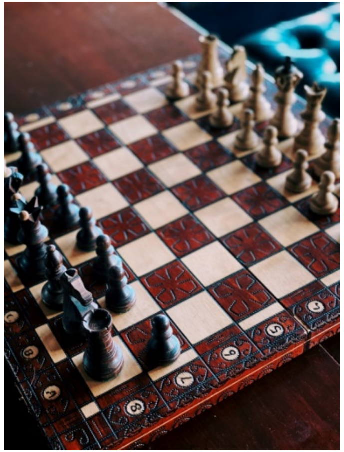
Alpha-Beta Pruning



How effective is **alpha-beta pruning**?

- Depends on the order of successors!
  - Best case, the # of nodes to search is O($b^{m/2}$)
  - Happens when each player's best move is the leftmost child.

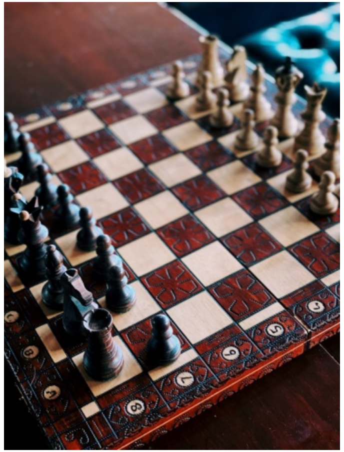
Alpha-Beta Pruning



How effective is **alpha-beta pruning**?

- Depends on the order of successors!
  - Best case, the # of nodes to search is O($b^{m/2}$)
  - Happens when each player's best move is the leftmost child.
  - The worst case is no pruning at all.

Alpha-Beta Pruning



How effective is **alpha-beta pruning**?

- Depends on the order of successors!
  - Best case, the # of nodes to search is $O(b^{m/2})$
  - Happens when each player's best move is the leftmost child.
  - The worst case is no pruning at all.

- In DeepBlue, the average branching factor was about 6 with alpha-beta instead of 35-40 without.

Minimax With Heuristics

*If $d = \infty$ then this pseudocode is equivalent to earlier minimax pseudocode. Check yourself!

Minimax With Heuristics

Note that long games may require huge computation

*If $d = \infty$ then this pseudocode is equivalent to earlier minimax pseudocode. Check yourself!

Minimax With Heuristics

Note that long games may require huge computation

- To deal with this: limit ***d*** for the search depth

*If $d = \infty$ then this pseudocode is equivalent to earlier minimax pseudocode. Check yourself!

Minimax With Heuristics

Note that long games may require huge computation

- To deal with this: limit ***d*** for the search depth
- **Q**: What to do at depth *d*, but no termination yet?
  - **A**: Use a heuristic evaluation function *e(x)*

*If $d = \infty$ then this pseudocode is equivalent to earlier minimax pseudocode. Check yourself!

Minimax With Heuristics

Note that long games may require huge computation

- To deal with this: limit ***d*** for the search depth
- **Q**: What to do at depth *d*, but no termination yet?
  - **A**: Use a heuristic evaluation function *e(x)*

function MINIMAX(x, d) **returns** an estimate of x's utility value
  inputs: x, current state in game
    d, an upper bound on the search depth
  if x is a terminal state **then return** Max's payoff at x
  else if $d = 0$ **then return** $e(x)$
  else if it is Max's move at x **then**
    return $\max\{$MINIMAX$(y, d-1) : y$ is a child of $x\}$
  else return $\min\{$MINIMAX$(y, d-1) : y$ is a child of $x\}$

Credit: Dana Nau

*If $d = \infty$ then this pseudocode is equivalent to earlier minimax pseudocode. Check yourself!

Heuristic Evaluation Functions

Heuristic Evaluation Functions

- e(x) can be any computable function of x; e.g. a weighted sum of features (like our linear models)

Heuristic Evaluation Functions

- e(x) can be any computable function of x; e.g. a weighted sum of features (like our linear models)

$$e(x) = w_1 f_1(x) + w_2 f_2(x) + \ldots + w_n f_n(x)$$

Heuristic Evaluation Functions

- e(x) can be any computable function of x; e.g. a weighted sum of features (like our linear models)

$$e(x) = w_1 f_1(x) + w_2 f_2(x) + \ldots + w_n f_n(x)$$

- Chess example: $f_i(x)$ = **difference** between number of white and black, with *i* ranging over piece types.
  - Set weights according to piece importance
  - E.g., 1(# white pawns - # black pawns) + 3(#white knights - # black knights)

Going Further

Going Further

- Monte Carlo tree search (MCTS)
  - Uses random sampling of the search space
  - Choose some children (heuristics to figure out #)
  - Record results, use for future play
  - Self-play

Going Further

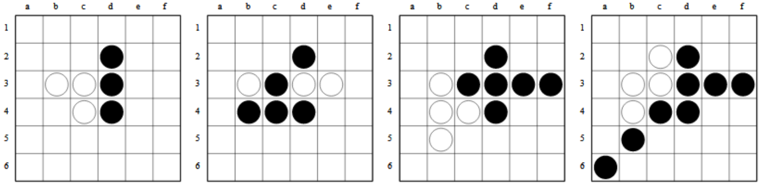
- Monte Carlo tree search (MCTS)
  - Uses random sampling of the search space
  - Choose some children (heuristics to figure out #)
  - Record results, use for future play
  - Self-play

- AlphaGo and other big results!

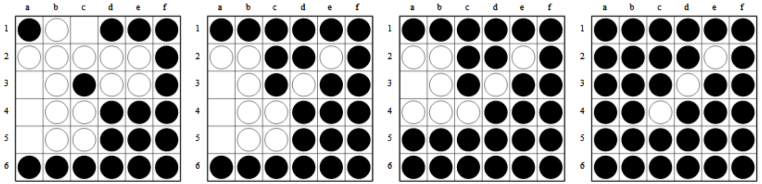
Going Further

- Monte Carlo tree search (MCTS)
  - Uses random sampling of the search space
  - Choose some children (heuristics to figure out #)
  - Record results, use for future play
  - Self-play

- AlphaGo and other big results!



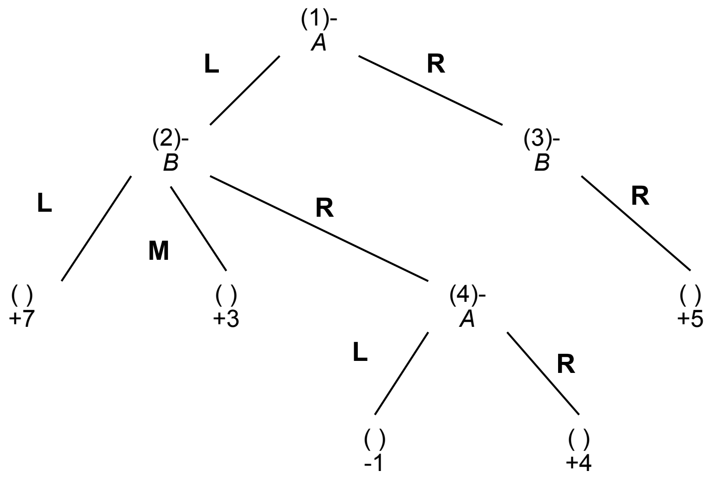
The agent (Black) learns to capture walls and corners in the early game



The agent (Black) learns to force passes in the late game

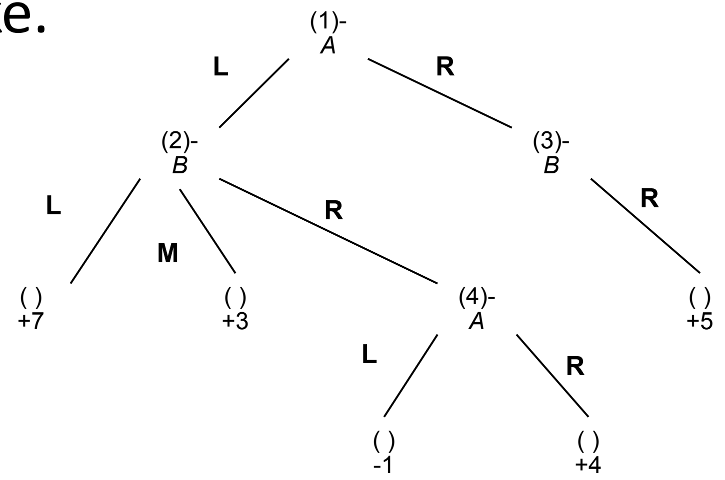
Credit: Surag Nair

From Extensive Form back to Normal Form Game



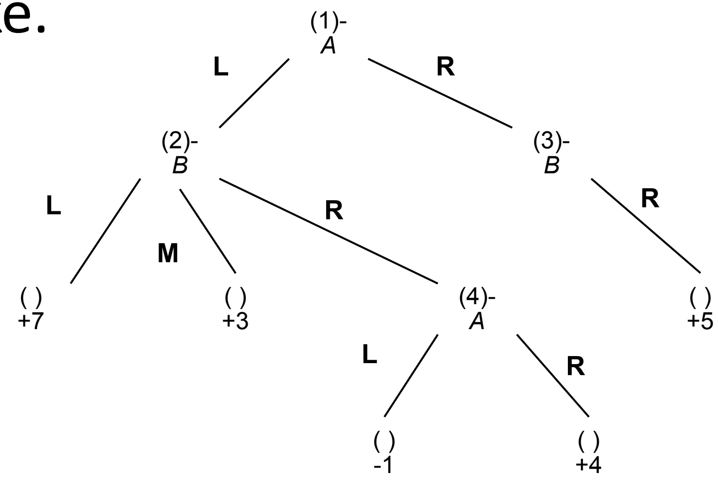
From Extensive Form back to Normal Form Game

- A pure strategy for a player is the mapping between all possible states the player can see, to the move the player would make.



From Extensive Form back to Normal Form Game

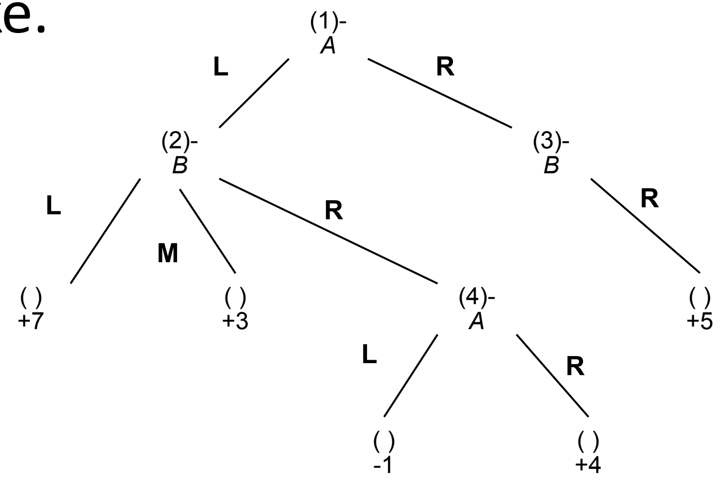
- A pure strategy for a player is the mapping between all possible states the player can see, to the move the player would make.
- Player A has 4 pure strategies:



From Extensive Form back to Normal Form Game

- A pure strategy for a player is the mapping between all possible states the player can see, to the move the player would make.
- Player A has 4 pure strategies:

A's strategy I: (1→L, 4→L)

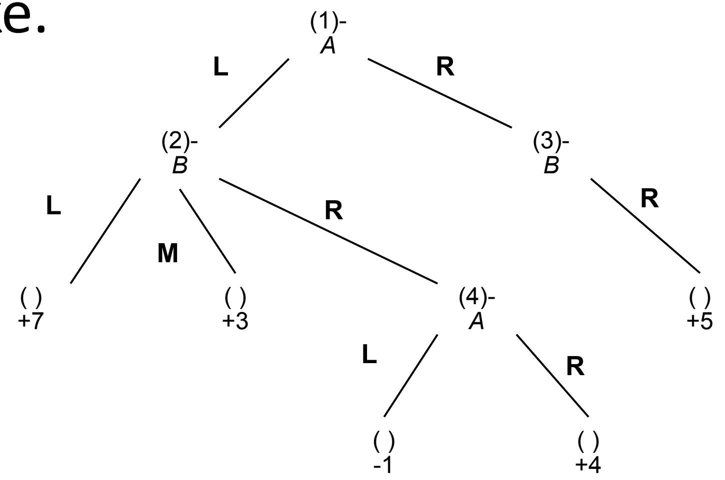


From Extensive Form back to Normal Form Game

- A pure strategy for a player is the mapping between all possible states the player can see, to the move the player would make.
- Player A has 4 pure strategies:

A’s strategy I: (1→L, 4→L)

A’s strategy II: (1→L, 4→R)



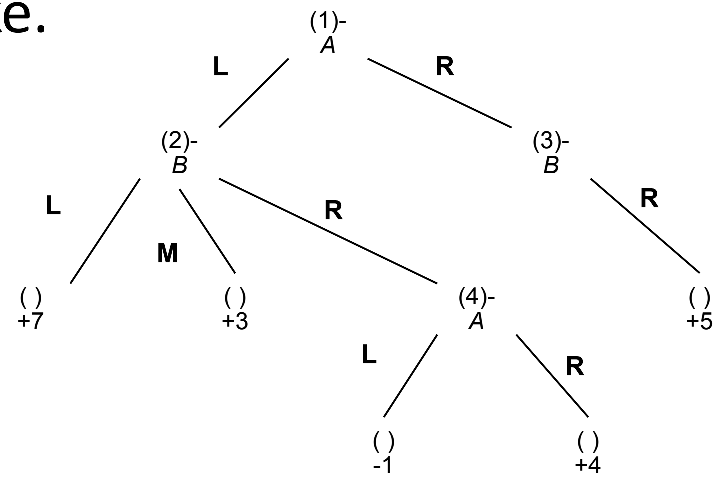
From Extensive Form back to Normal Form Game

- A pure strategy for a player is the mapping between all possible states the player can see, to the move the player would make.
- Player A has 4 pure strategies:

A's strategy I: (1→L, 4→L)

A's strategy II: (1→L, 4→R)

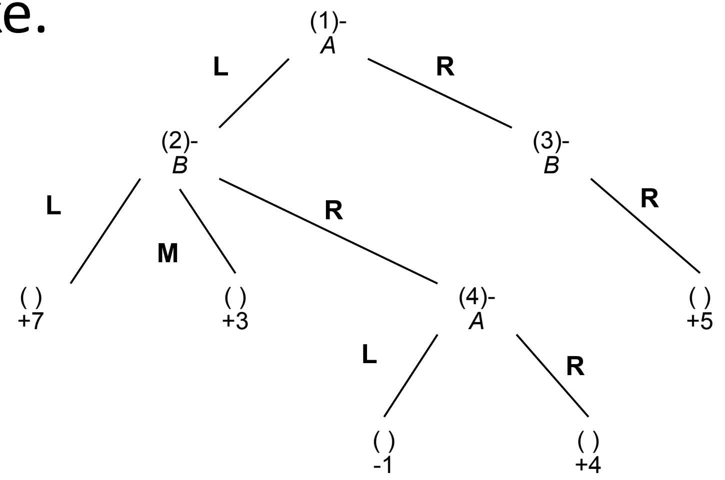
A's strategy III: (1→R, 4→L)



From Extensive Form back to Normal Form Game

- A pure strategy for a player is the mapping between all possible states the player can see, to the move the player would make.
- Player A has 4 pure strategies:

A's strategy I: (1→L, 4→L)
A's strategy II: (1→L, 4→R)
A's strategy III: (1→R, 4→L)
A's strategy IV: (1→R, 4→R)



From Extensive Form back to Normal Form Game

- A pure strategy for a player is the mapping between all possible states the player can see, to the move the player would make.
- Player A has 4 pure strategies:

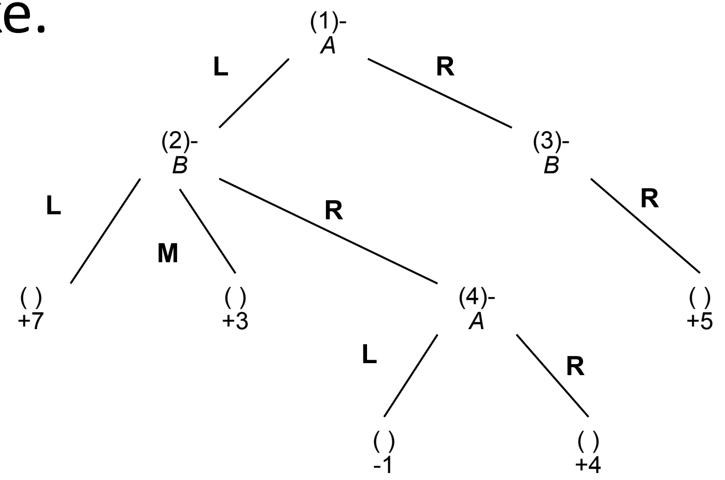
  A’s strategy I: (1→L, 4→L)

  A’s strategy II: (1→L, 4→R)

  A’s strategy III: (1→R, 4→L)

  A’s strategy IV: (1→R, 4→R)

- Player B has 3 pure strategies:



From Extensive Form back to Normal Form Game

- A pure strategy for a player is the mapping between all possible states the player can see, to the move the player would make.
- Player A has 4 pure strategies:

A’s strategy I: (1→L, 4→L)

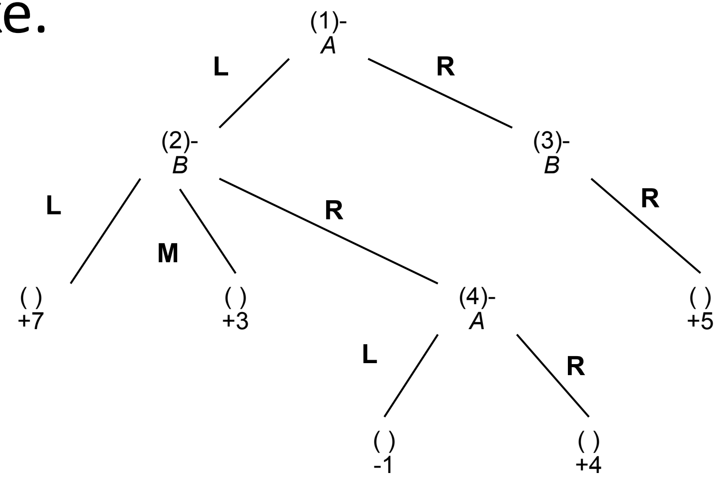
A’s strategy II: (1→L, 4→R)

A’s strategy III: (1→R, 4→L)

A’s strategy IV: (1→R, 4→R)

- Player B has 3 pure strategies:

B’s strategy I: (2→L, 3→R)



From Extensive Form back to Normal Form Game

- A pure strategy for a player is the mapping between all possible states the player can see, to the move the player would make.
- Player A has 4 pure strategies:

A's strategy I: (1→L, 4→L)

A's strategy II: (1→L, 4→R)

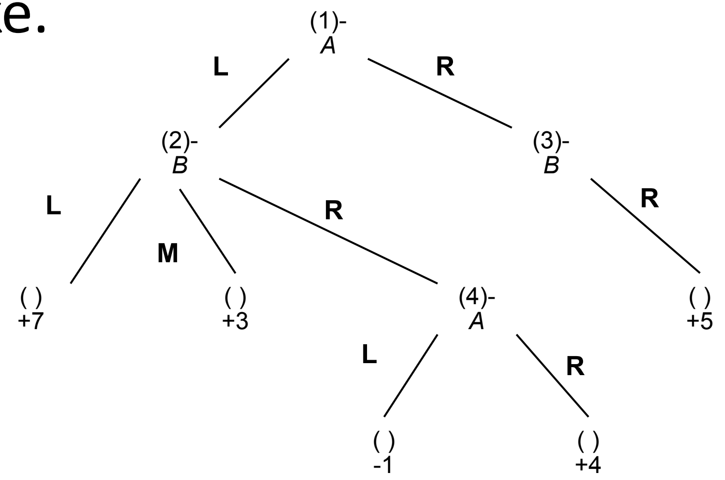
A's strategy III: (1→R, 4→L)

A's strategy IV: (1→R, 4→R)

- Player B has 3 pure strategies:

B's strategy I: (2→L, 3→R)

B's strategy II: (2→M, 3→R)



From Extensive Form back to Normal Form Game

- A pure strategy for a player is the mapping between all possible states the player can see, to the move the player would make.
- Player A has 4 pure strategies:

  A's strategy I: (1→L, 4→L)

  A's strategy II: (1→L, 4→R)

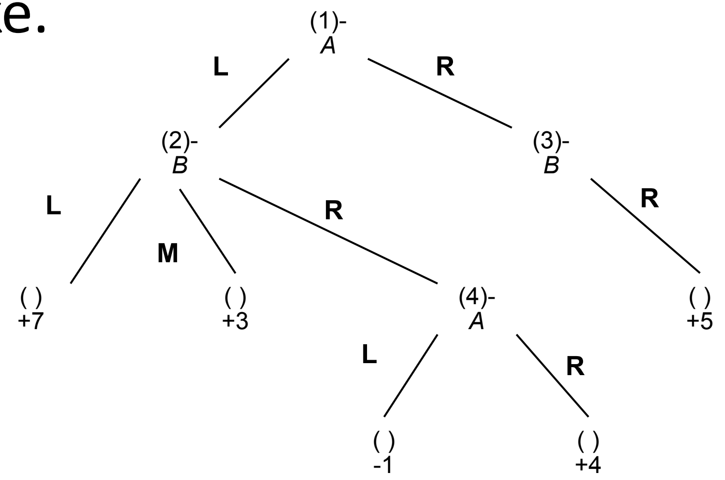
  A's strategy III: (1→R, 4→L)

  A's strategy IV: (1→R, 4→R)
- Player B has 3 pure strategies:

  B's strategy I: (2→L, 3→R)

  B's strategy II: (2→M, 3→R)

  B's strategy III: (2→R, 3→R)



From Extensive Form back to Normal Form Game

- A pure strategy for a player is the mapping between all possible states the player can see, to the move the player would make.
- Player A has 4 pure strategies:

  A's strategy I: ($1\rightarrow L$, $4\rightarrow L$)

  A's strategy II: ($1\rightarrow L$, $4\rightarrow R$)

  A's strategy III: ($1\rightarrow R$, $4\rightarrow L$)

  A's strategy IV: ($1\rightarrow R$, $4\rightarrow R$)

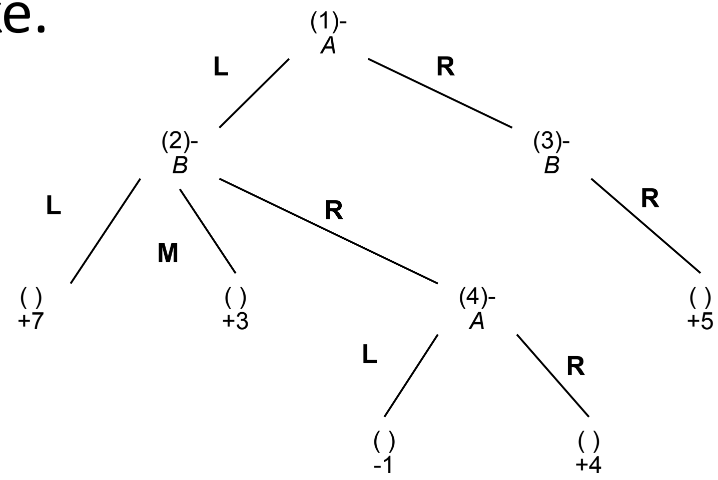
- Player B has 3 pure strategies:

  B's strategy I: ($2\rightarrow L$, $3\rightarrow R$)

  B's strategy II: ($2\rightarrow M$, $3\rightarrow R$)

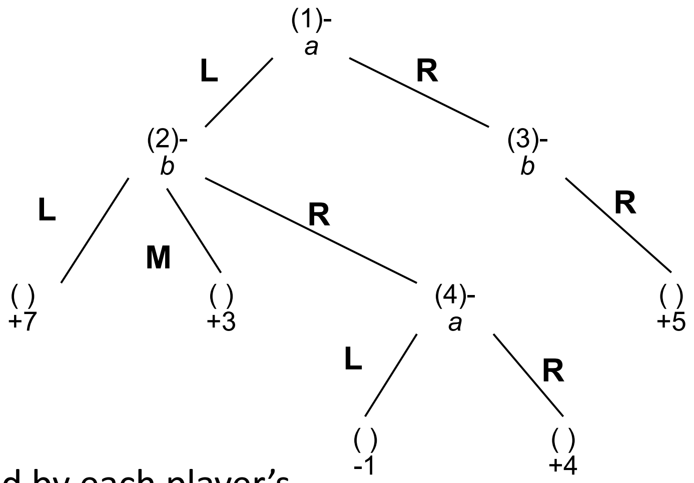
  B's strategy III: ($2\rightarrow R$, $3\rightarrow R$)

- How many pure strategies if each player can see *N* states, and has *b* moves at each state?



Matrix Normal Form of games

A's strategy I: (1→L, 4→L)
A's strategy II: (1→L, 4→R)
A's strategy III: (1→R, 4→L)
A's strategy IV: (1→R, 4→R)
B's strategy I: (2→L, 3→R)
B's strategy II: (2→M, 3→R)
B's strategy III: (2→R, 3→R)

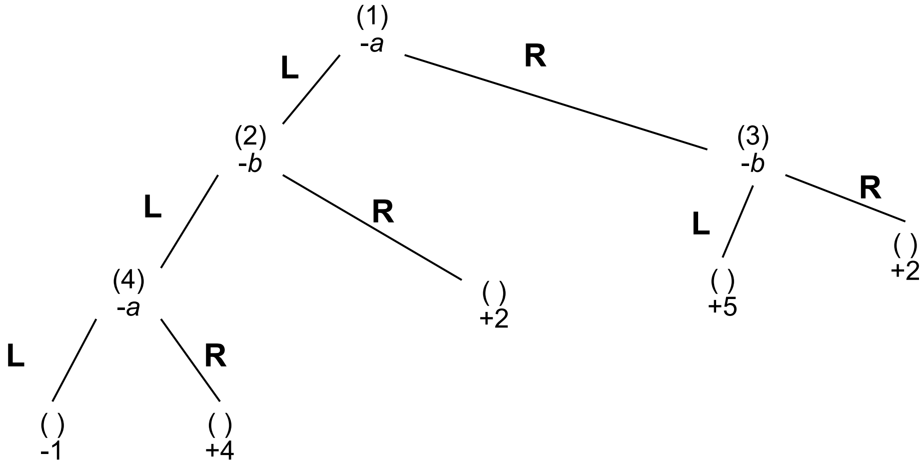


The matrix normal form is the game value matrix indexed by each player's strategies.

	B-I	B-II	B-III
A-I	7	3	-1
A-II	7	3	4
A-III	5	5	5
A-IV	5	5	5

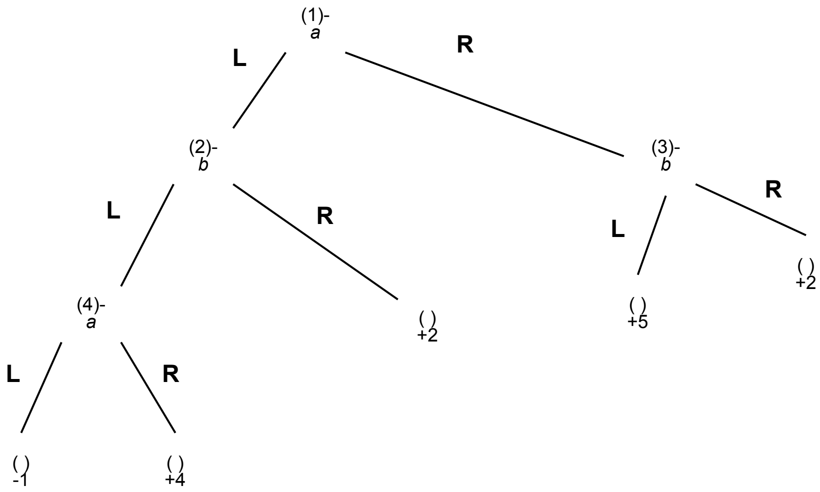
The matrix encodes every outcome of the game! The rules etc. are no longer needed.

Another example of normal form



- How many pure strategies does A have?
- How many does B have?
- What is the matrix form of this game?

Matrix normal form example



	B-I	B-II	B-III	B-IV
A-I	-1	-1	2	2
A-II	4	4	2	2
A-III	5	2	5	2
A-IV	5	2	5	2

- How many pure strategies does A have? 4

A-I (1→L, 4→L) A-II (1→L,4→R) A-III (1→R,4→L) A-IV (1→R, 4→R)

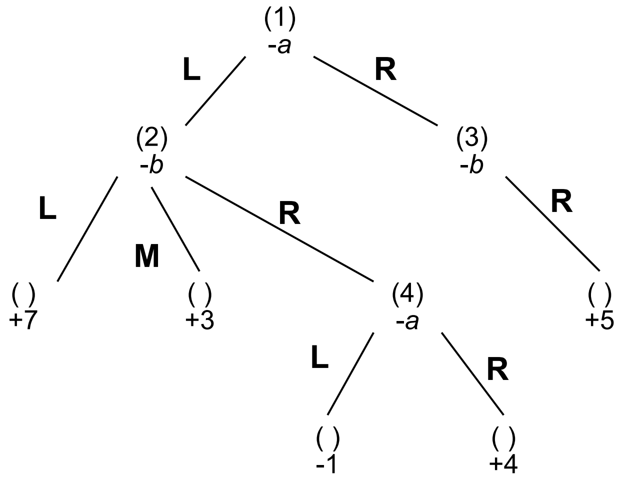
- How many does B have? 4

B-I (2→L, 3→L) B-II (2→L,3→R) B-III (2→R,3→L) B-IV (2→R, 3→R)

- What is the matrix form of this game?

Minimax in Matrix Normal Form

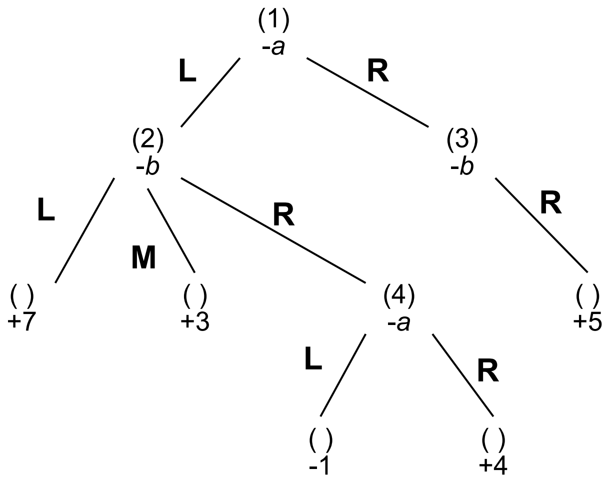
- Player A: for each strategy, consider all B's counter strategies (a row in the matrix), find the minimum value in that row. Pick the row with the maximum minimum value.
- Here maximin=5



	B-I	B-II	B-III
A-I	7	3	-1
A-II	7	3	4
A-III	5	5	5
A-IV	5	5	5

Minimax in Matrix Normal Form

- Player B: find the maximum value in each column. Pick the column with the minimum maximum value.
- Here minimax = 5



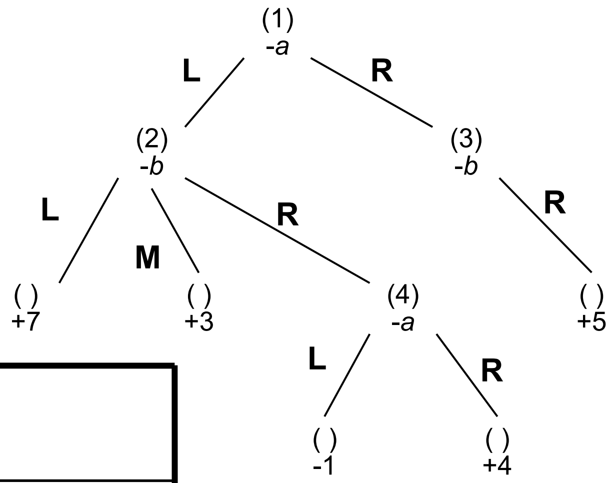
Fundamental game theory result (proved by von Neumann):

In a 2-player, zero-sum game of perfect information (sequential moves), Minimax==Maximin. And there always exists an optimal pure strategy for each player.

	B-I	B-II	B-III
A-I	7	3	-1
A-II	7	3	4
A-III	5	5	5
A-IV	5	5	5

Minimax in Matrix Normal Form

- We can also check for mutual best responses



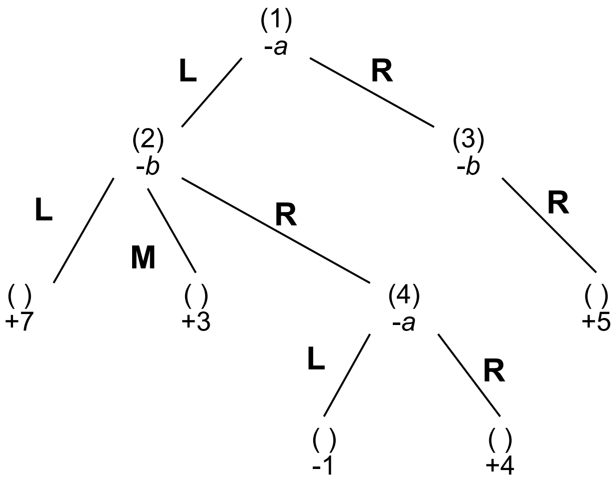
	B-I	B-II	B-III
A-I	7	3	-1
A-II	7	3	4
A-III	5	5	5
A-IV	5	5	5

Minimax in Matrix Normal Form

Interestingly, A can tell B in advance what strategy A will use (the maximin), and this information will not help B!
Similarly B can tell A what strategy B will use.
In fact A knows what B's strategy will be.
And B knows A's too.
And A knows that B knows
…
The game is at an equilibrium

e

player.



	B-I	B-II	B-III
A-I	7	3	-1
A-II	7	3	4
A-III	5	5	5
A-IV	5	5	5