CS 540 Introduction to Artificial Intelligence
Game I

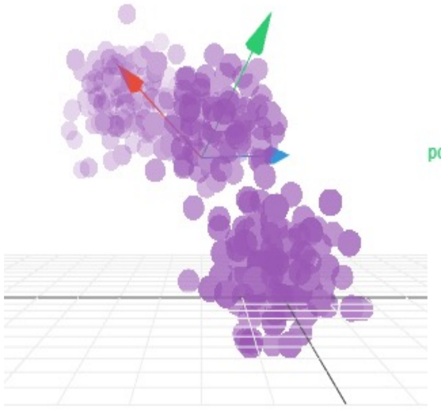
Yingyu Liang
University of Wisconsin-Madison
Nov 23, 2021

Based on slides by Fred Sala

Outline

- Intro to game theory
  - Characterize games by various properties
- Sequential games
  - Game trees, game-theoretic/minimax value, minimax algo
- Improving our search
  - Using heuristics

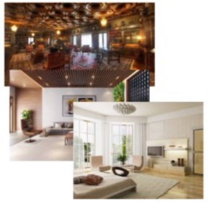
So Far in The Course



Victor Powell

We looked at techniques:

- **Unsupervised:** See data, do something with it. Unstructured.
- **Supervised:** Train a model to make predictions. More structure.
  - Training: as taking actions to get a reward
- **Games**: Much more structure.

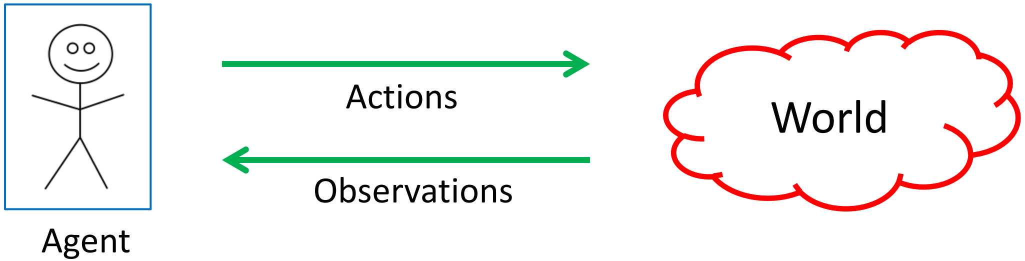


indoor

outdoor

More General Model

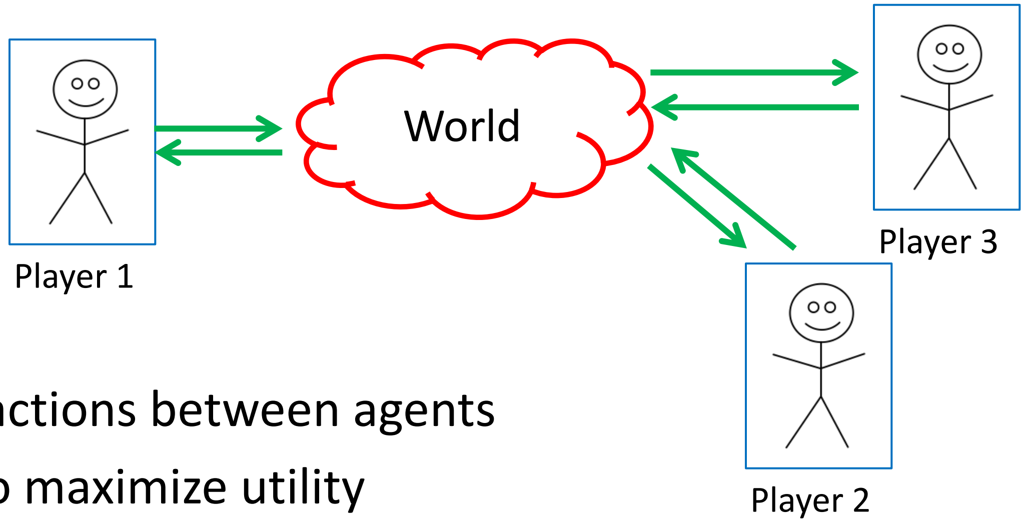
Suppose we have an **agent** **interacting** with the **world**



- Agent receives a reward based on state of the world
  - **Goal**: maximize reward / utility **($$$)**
  - Note: now **data** consists of actions & observations
  - Setup for decision theory, reinforcement learning, planning

Games: Multiple Agents

Games setup: **multiple** agents



- Now: interactions between agents
- Still want to maximize utility
- **Strategic** decision making.

Modeling Games: Properties

Let's work through **properties** of games

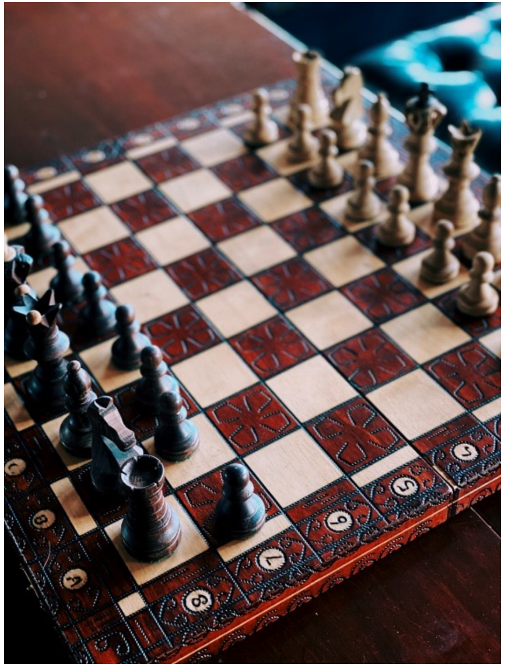
- **Number** of agents/players
- State & action spaces: **discrete** or **continuous**
- **Finite** or **infinite**
- **Deterministic** or **random**
- **Sum**: zero or positive or negative
- **Sequential** or **simultaneous**

Wiki

Property 1: **Number** of players

Pretty clear idea: 1 or more players

- Usually interested in ≥ 2 players
- Typically a finite number of players



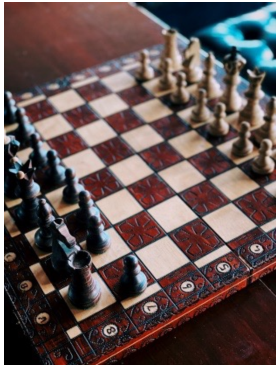
Property 2: Discrete or Continuous

Let's work through **properties** of games

- Recall the **world**. It is in a particular state, from a set of states
- Similarly, the actions the player takes are from an action space
- How big are these spaces? Finite, countable, uncountable?

Property 3: Finite or Infinite

Let's work through **properties** of games



- Most real-world games **finite**
- Lots of single-turn games; end immediately
  - Ex: rock/paper/scissors
- Other games' rules (state & action spaces) enforce termination
  - Ex: chess under FIDE rules ends in at most 8848 moves
- **Infinite example:** pick integers. First player to play a 5 loses

Property 4: Deterministic or Random

Let's work through **properties** of games

- Is there **chance** in the game?
- Note: randomness enters in different ways

Property 5: Sums

Let’s work through **properties** of games

- **Sum**: zero or positive or negative
- Zero sum: for one player to win, the other has to lose
  - No “value” created

Blue / Red	*A*	*B*	*C*
1	−30 / 30	10 / −10	−20 / 20
2	10 / −10	−20 / 20	20 / −20

- Can have other types of games: positive sum, negative sum.
  - Example: prisoner’s dilemma

Property 6: Sequential or Simultaneous

Let's work through **properties** of games

- Sequential or simultaneous
- Simultaneous: all players take action at the same time
- Sequential: take turns

- Simultaneous: players do not have information of others' moves. Ex: **RPS**
- Sequential: may or may not have **perfect information** (knowledge of all moves so far)

Examples

Let's apply this to examples:

1. Chess: **2-player, discrete, finite, deterministic, zero-sum, sequential (perfect information)**

2. RPS: **2-player, discrete, finite, deterministic, zero-sum, simultaneous**

3. Mario Kart: **4-player, continuous, infinite (?), random, zero-sum, simultaneous**

Another Example: Prisoner's Dilemma

Famous example from the '50s.

Two prisoners A & B. Can choose to betray the other or not.

- A and B both betray, each of them serves two years in prison
- One betrays, the other doesn't: betrayer free, other three years
- Both do not betray: one year each

Properties: **2-player, discrete, finite, deterministic, negative-sum, simultaneous**

Why Do These Properties Matter?

Categorize games in different groups

- Can focus on understanding/analyzing/“solving” particular groups
- **Abstract** away details and see common patterns
- Understand how to produce a “good” overall outcome

How Does it Connect To Learning?

Obviously, learn how to play effectively

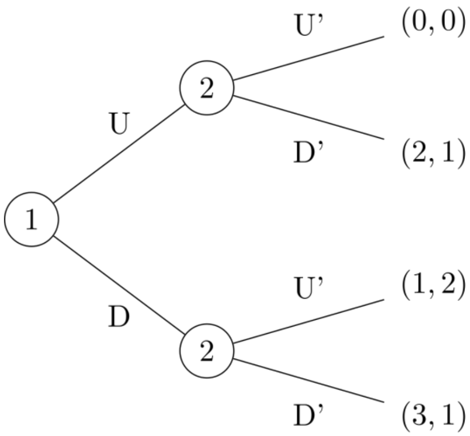
Also: suppose the players don't know something

- **Ex**: the reward / utility function is not known
- Common for real-world situations
  - How do we choose actions?
- Model the reward function and **learn it**
  - Try out actions and observe the rewards

Sequential Games

Games with multiple moves

- Represent with a **tree**
- Perform search over the tree



Wiki

II-Nim: Example Sequential 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, the score is **+1**; otherwise **-1**
- Min's score is –Max's
- 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

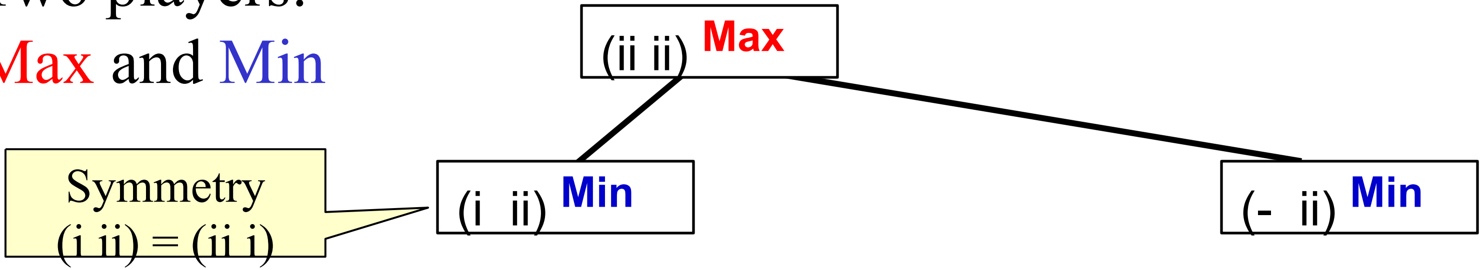
who is to move at this state

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

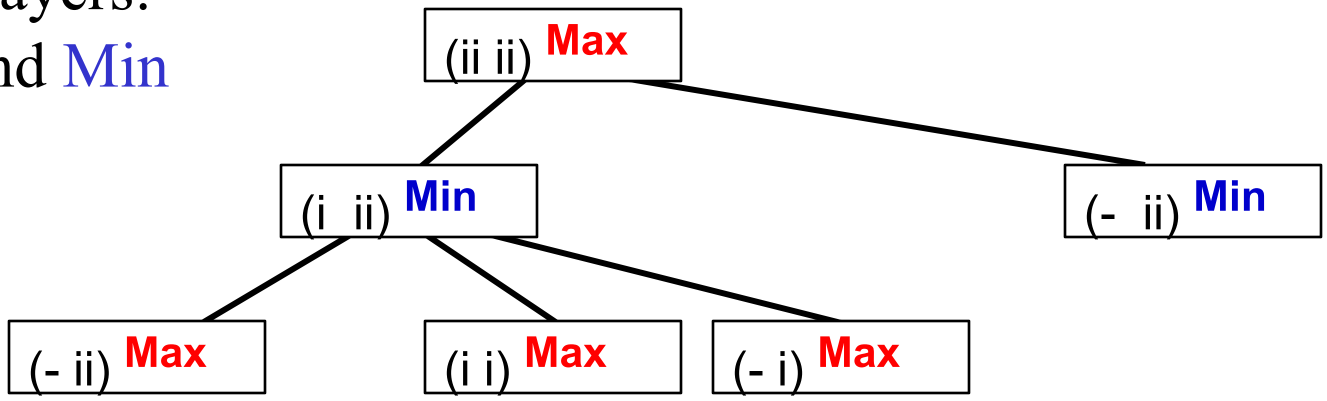
Two players:
Max and Min



Max wants the largest score
Min wants the smallest score

Game tree for II-Nim

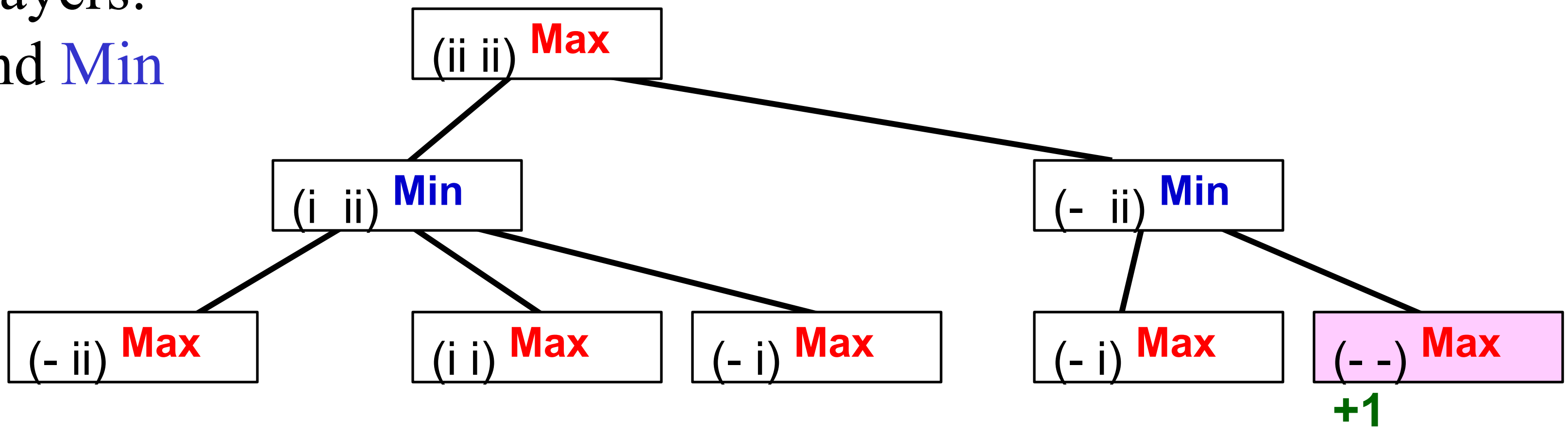
Two players:
Max and Min



Max wants the largest score
Min wants the smallest score

Game tree for II-Nim

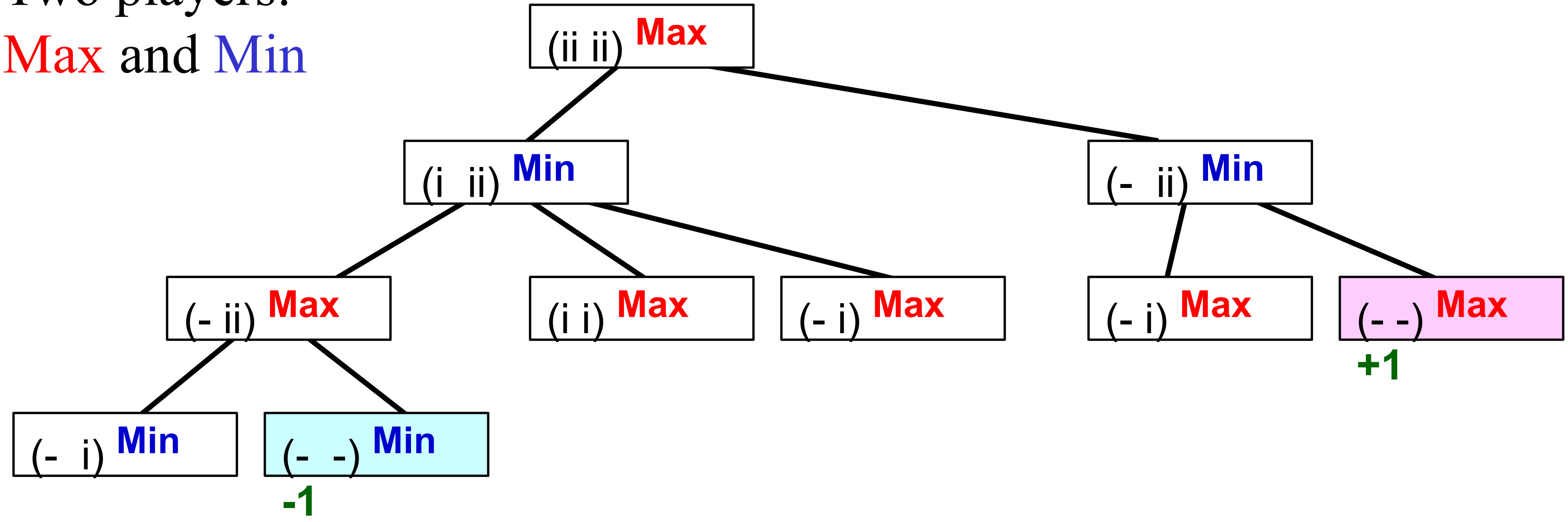
Two players:
Max and Min



Max wants the largest score
Min wants the smallest score

Game tree for II-Nim

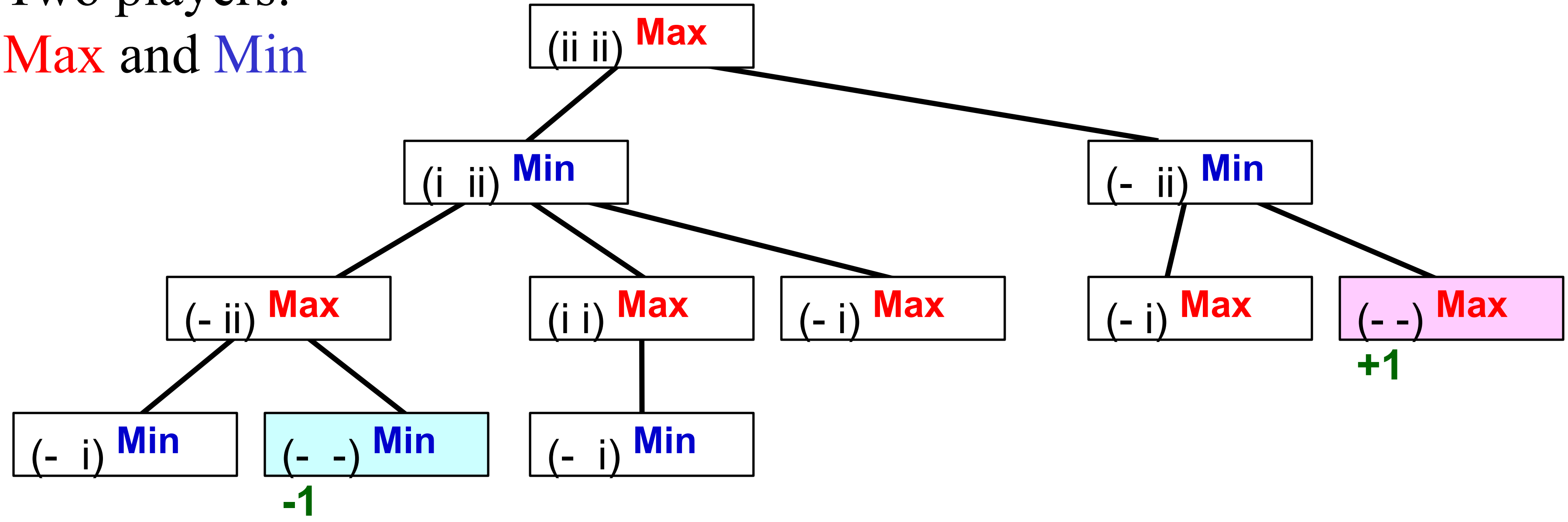
Two players:
Max and Min



Max wants the largest score
Min wants the smallest score

Game tree for II-Nim

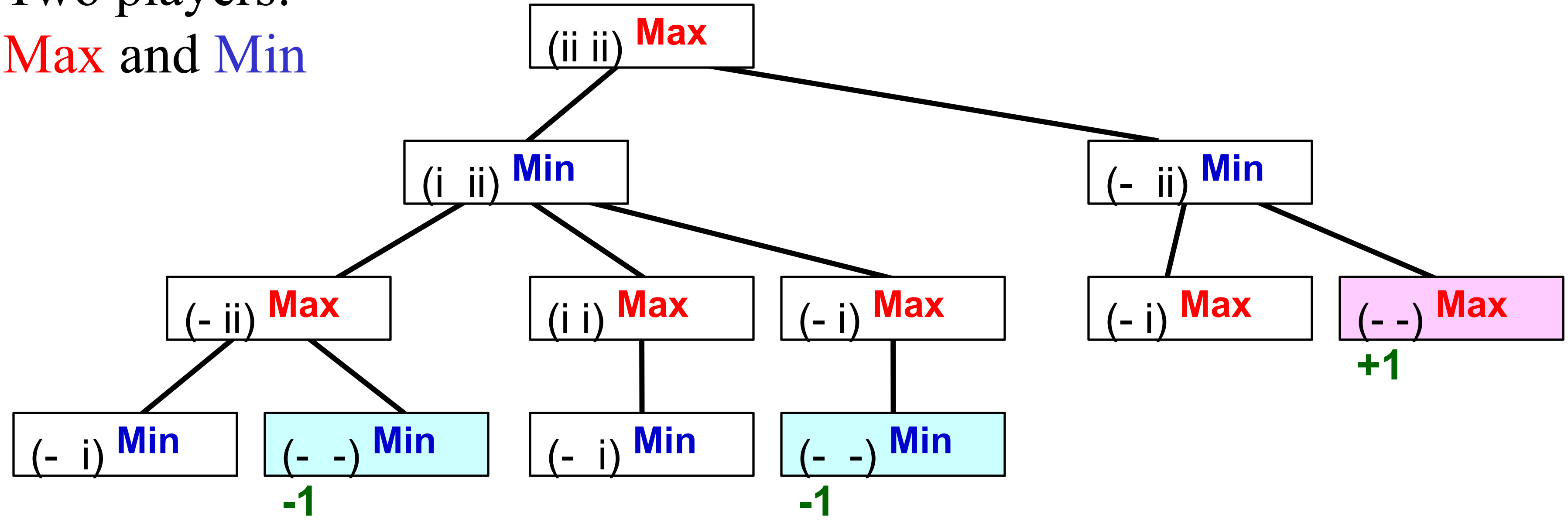
Two players:
Max and Min



Max wants the largest score
Min wants the smallest score

Game tree for II-Nim

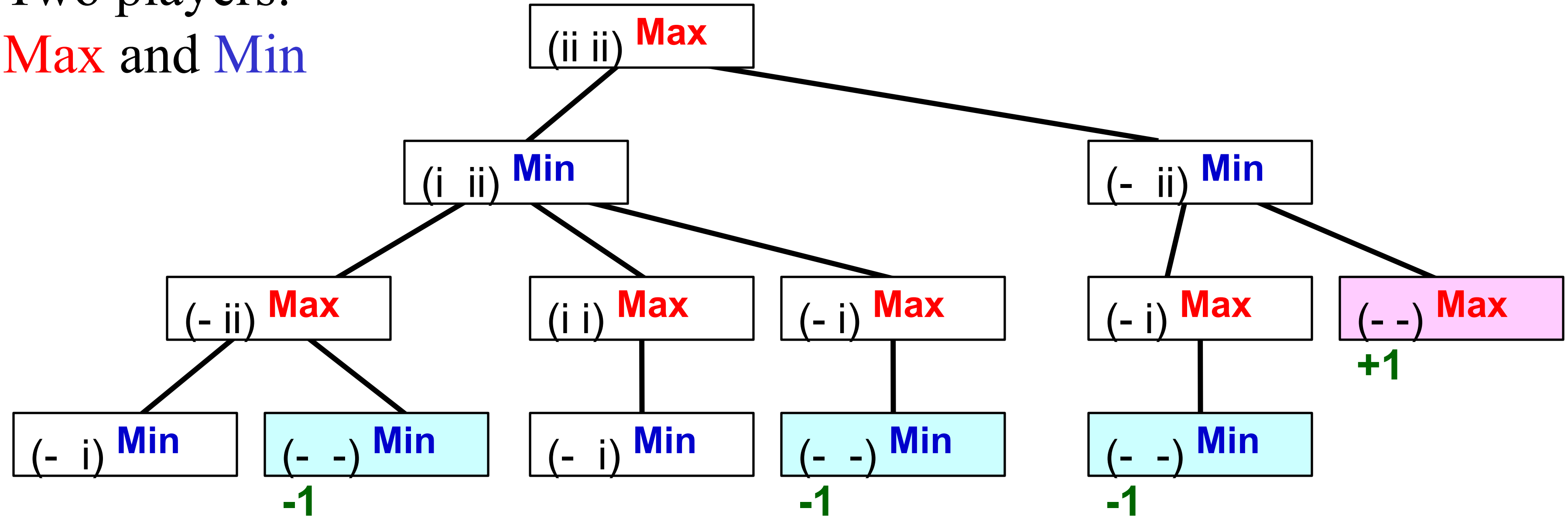
Two players:
Max and Min



Max wants the largest score
Min wants the smallest score

Game tree for II-Nim

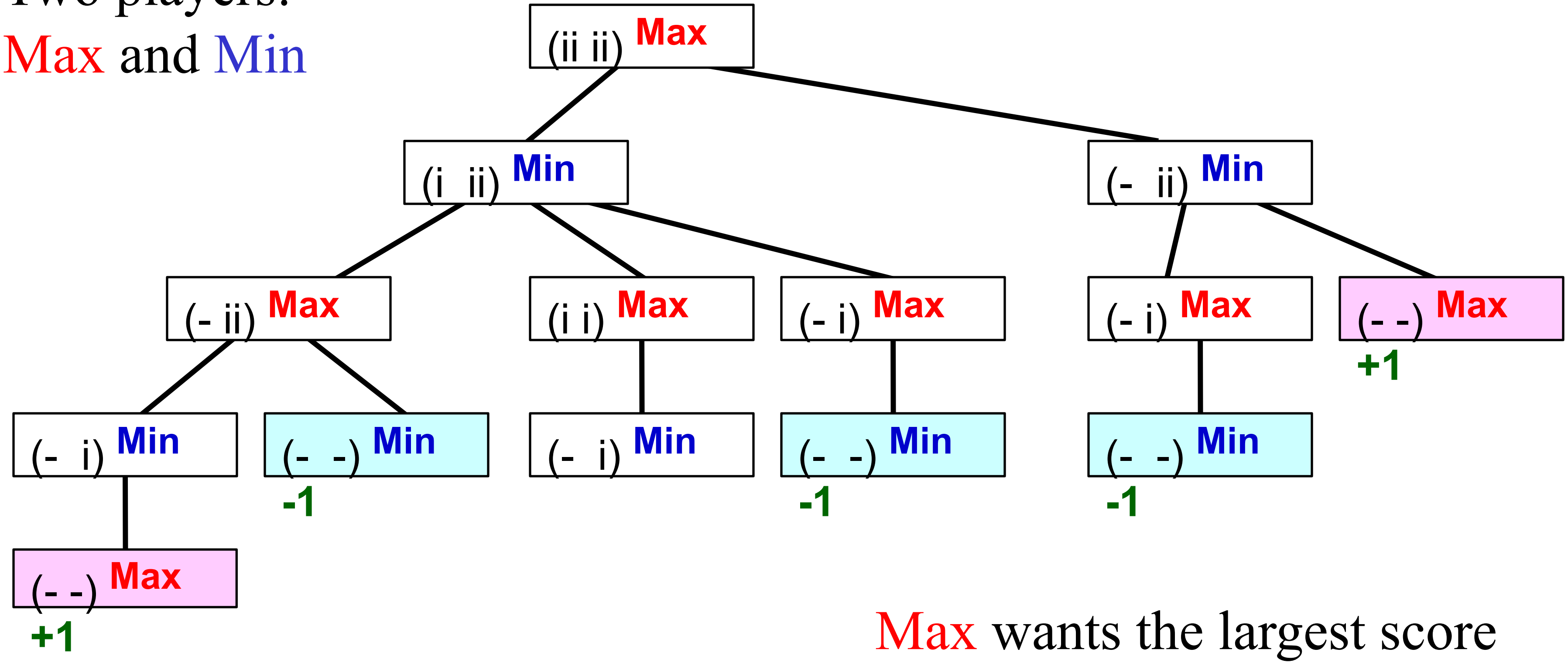
Two players:
Max and Min



Max wants the largest score
Min wants the smallest score

Game tree for II-Nim

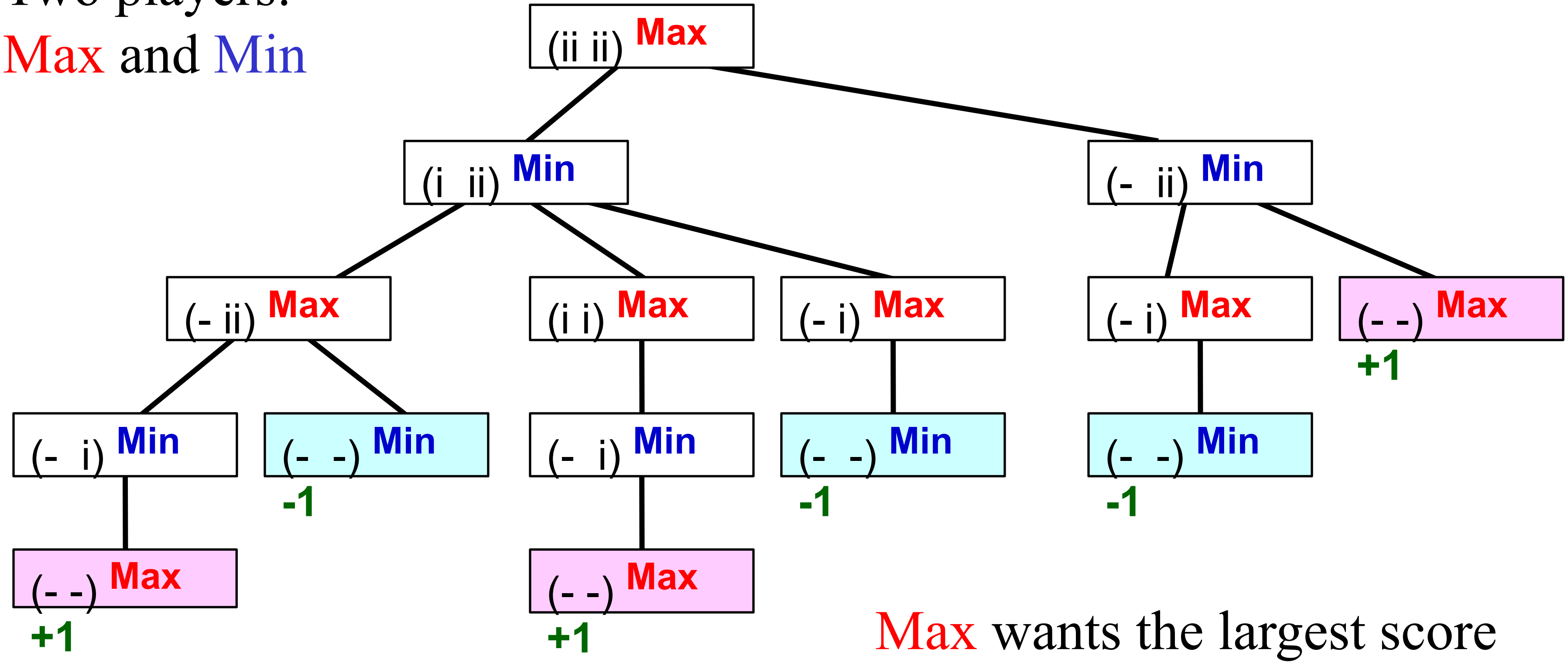
Two players:
Max and Min



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

Minimax Value

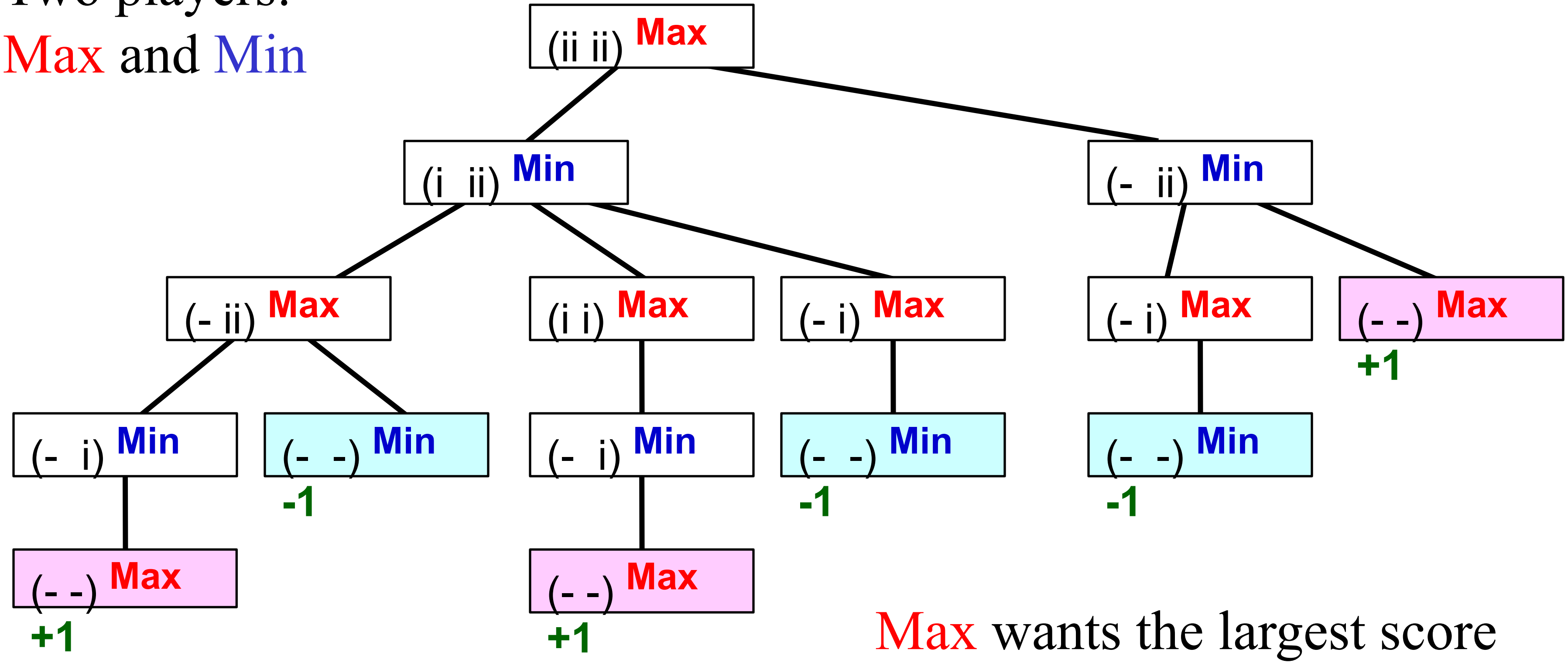
Also called **game-theoretic value**.

- Score of terminal node if both players play optimally.
- Computed bottom up; basically search

- Let’s see this for example game

Game tree for II-Nim

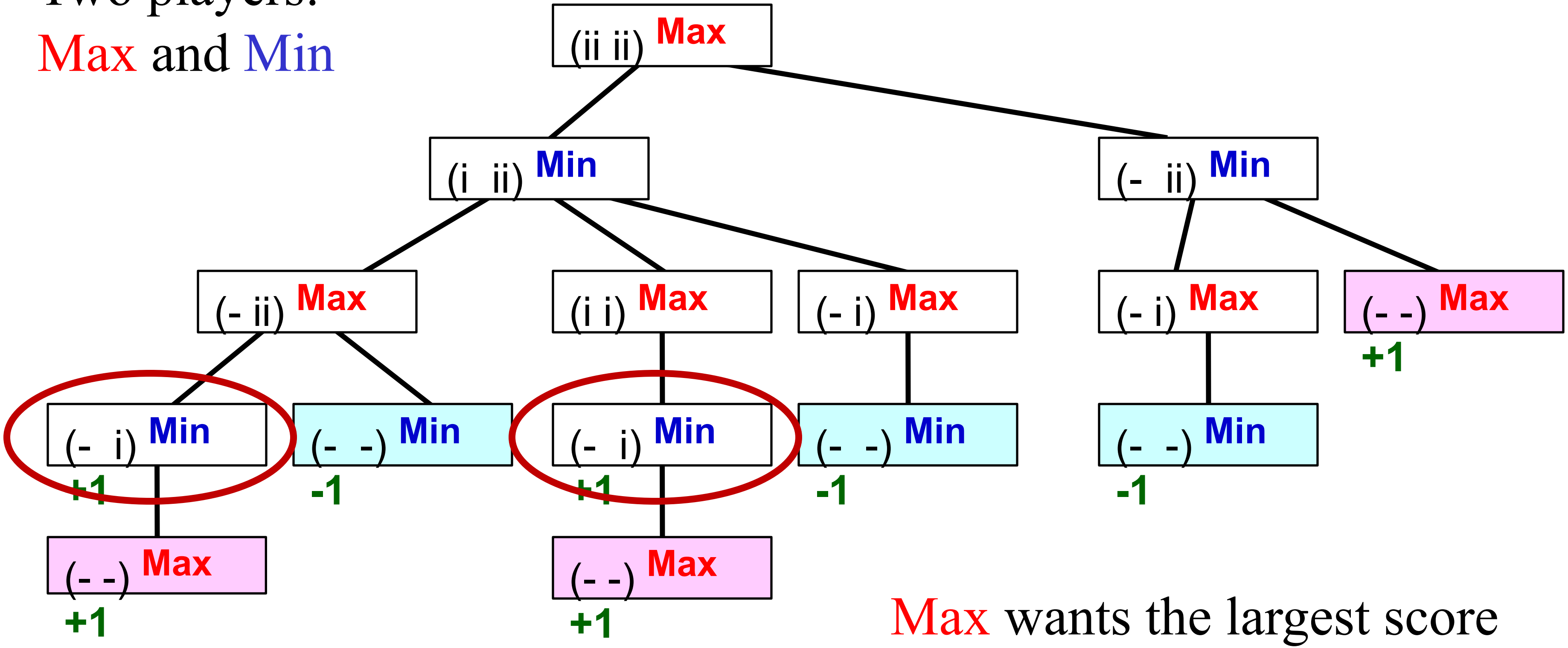
Two players:
Max and Min



Max wants the largest score
Min wants the smallest score

Game tree for II-Nim

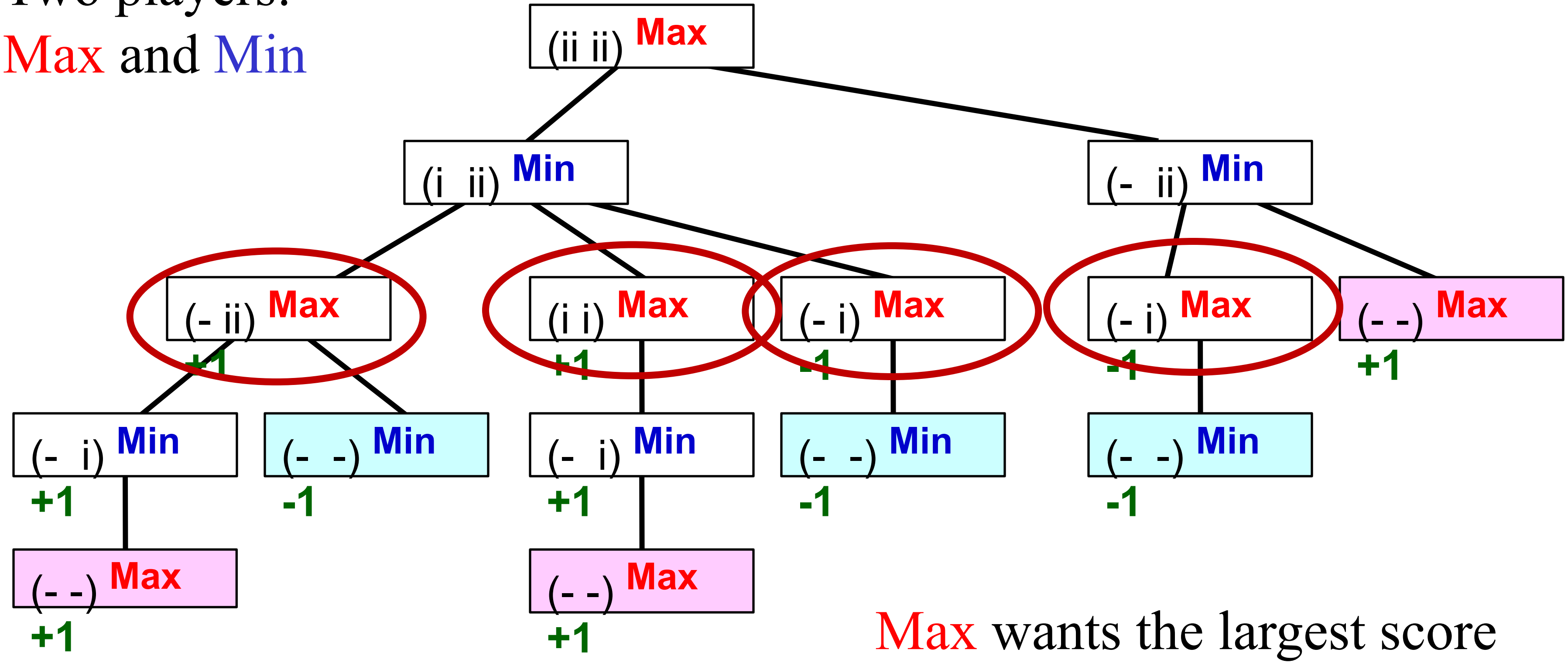
Two players:
Max and Min



Max wants the largest score
Min wants the smallest score

Game tree for II-Nim

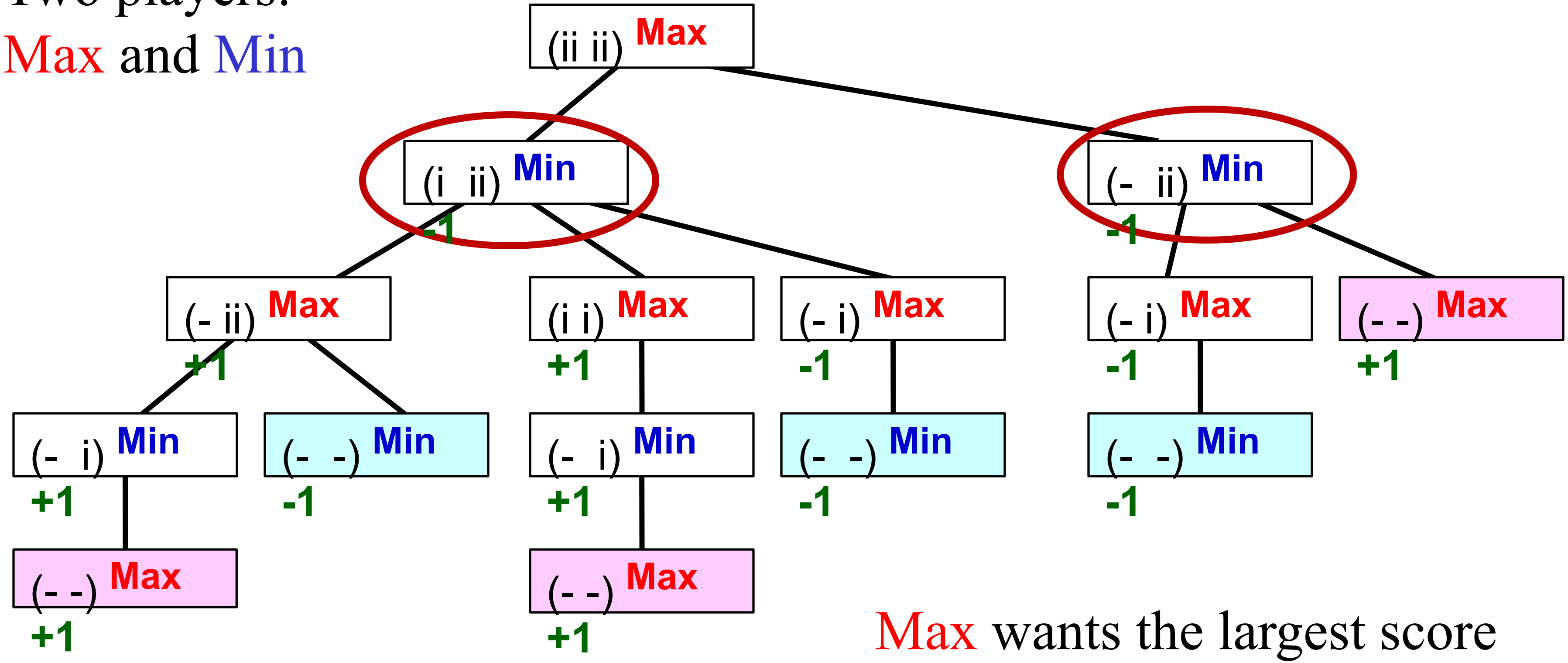
Two players:
Max and Min



Max wants the largest score
Min wants the smallest score

Game tree for II-Nim

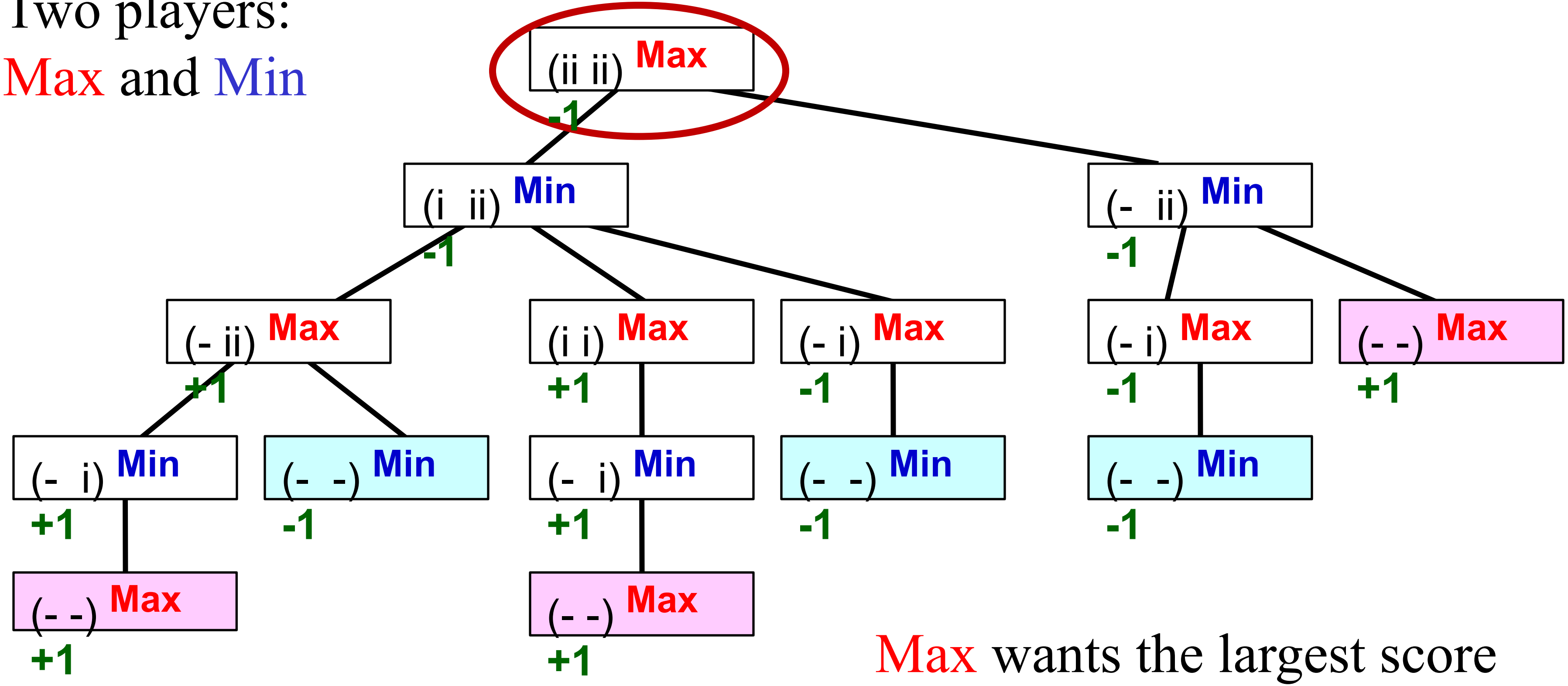
Two players:
Max and Min



Max wants the largest score
Min wants the smallest score

Game tree for II-Nim

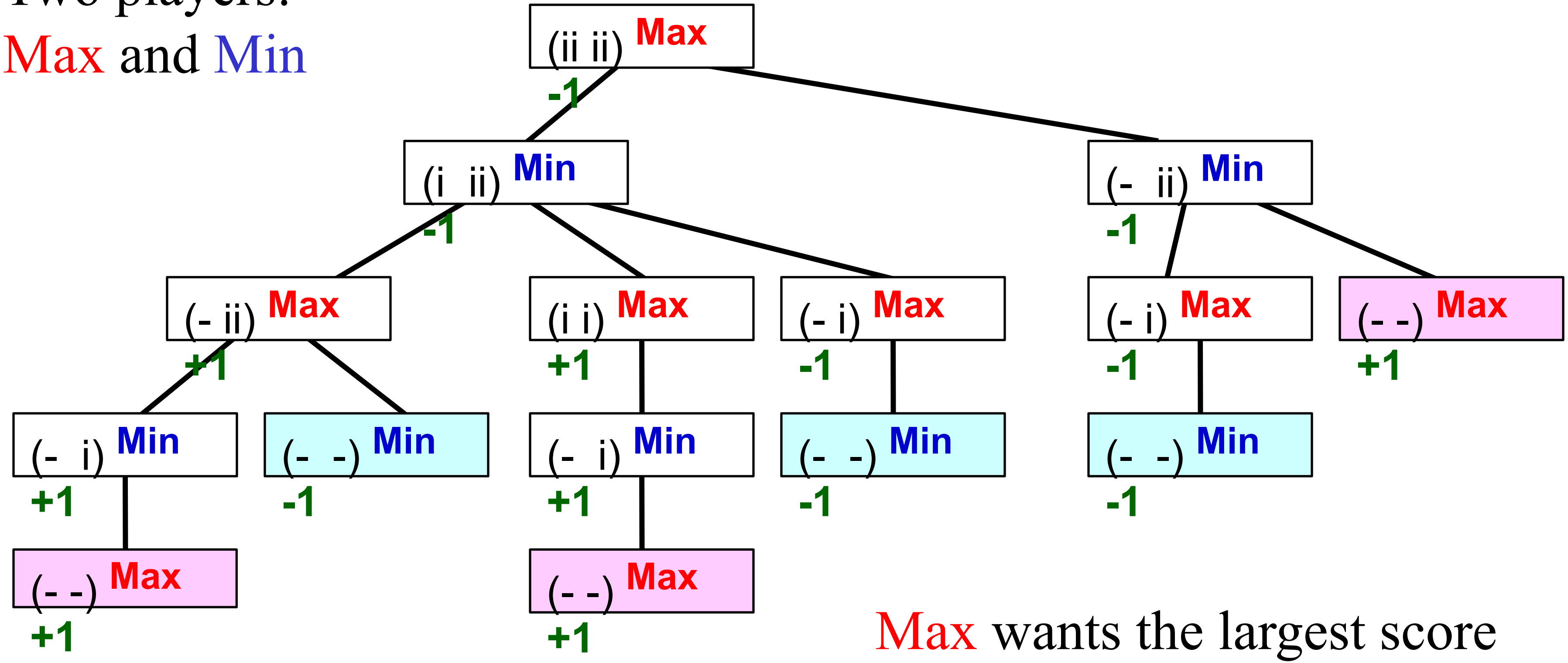
Two players:
Max and Min



Max wants the largest score
Min wants the smallest score

Game tree for II-Nim

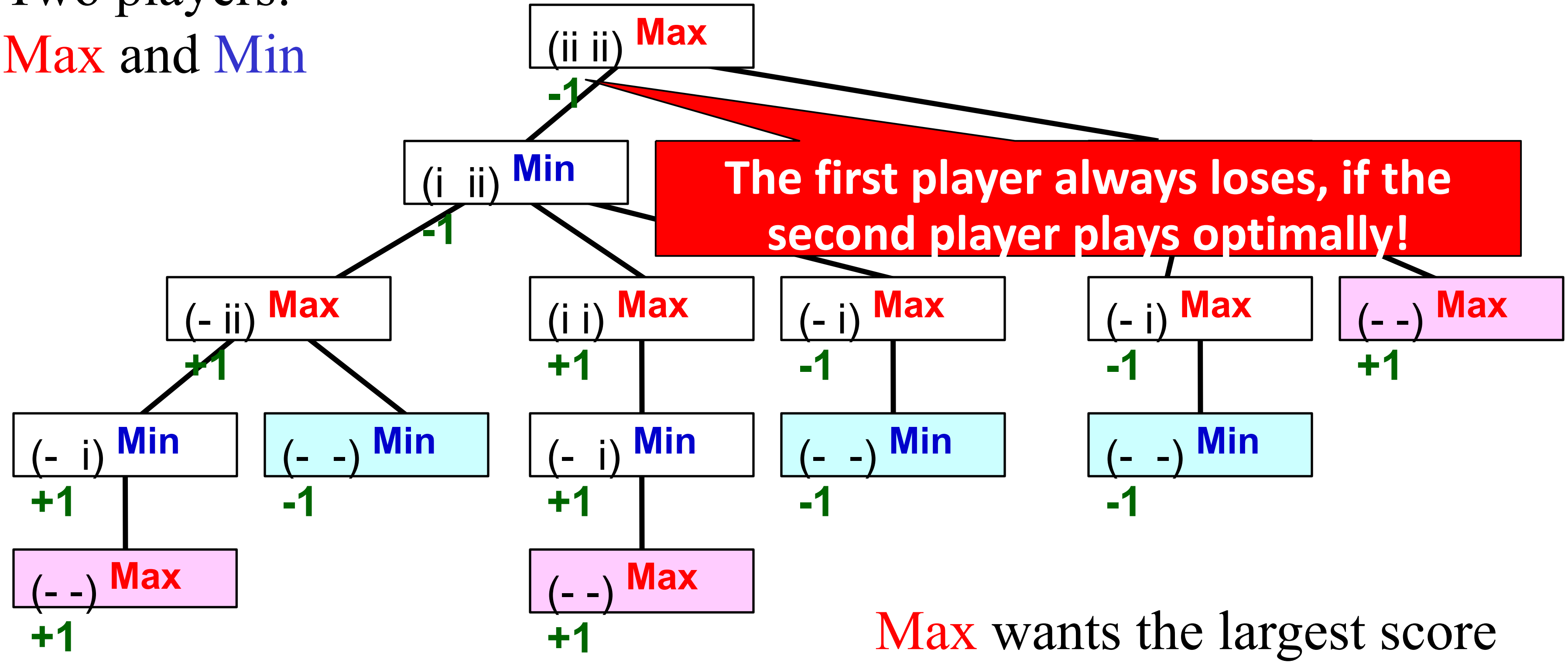
Two players:
Max and Min



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

- 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
```

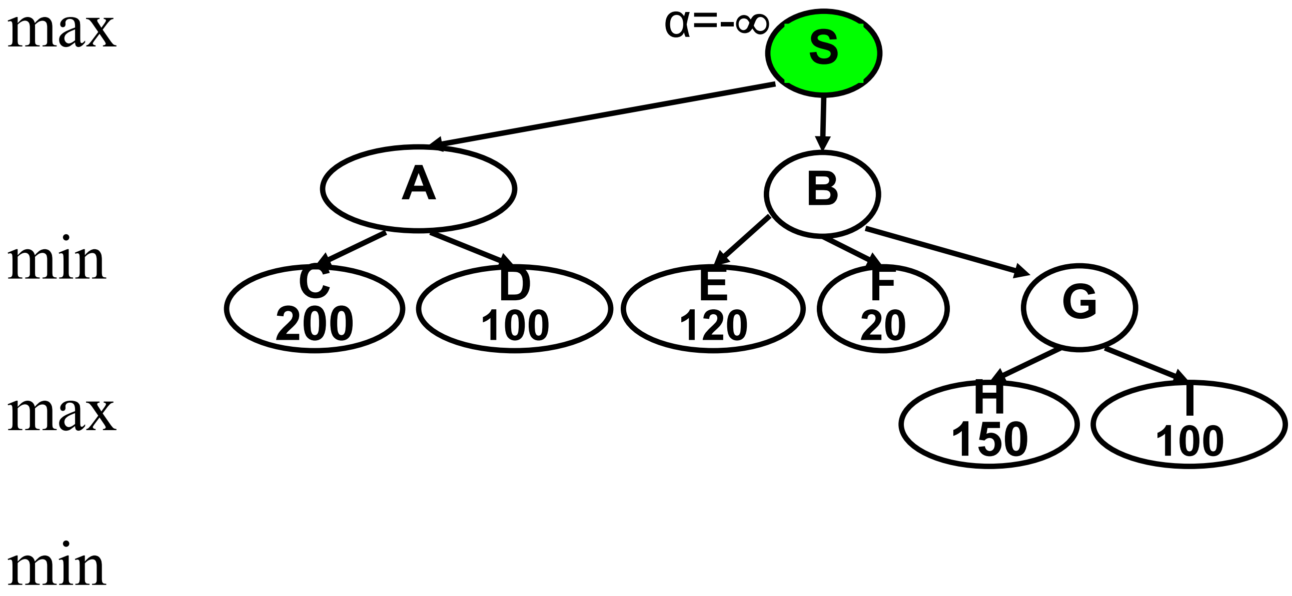
Time complexity?

- $O(b^m)$

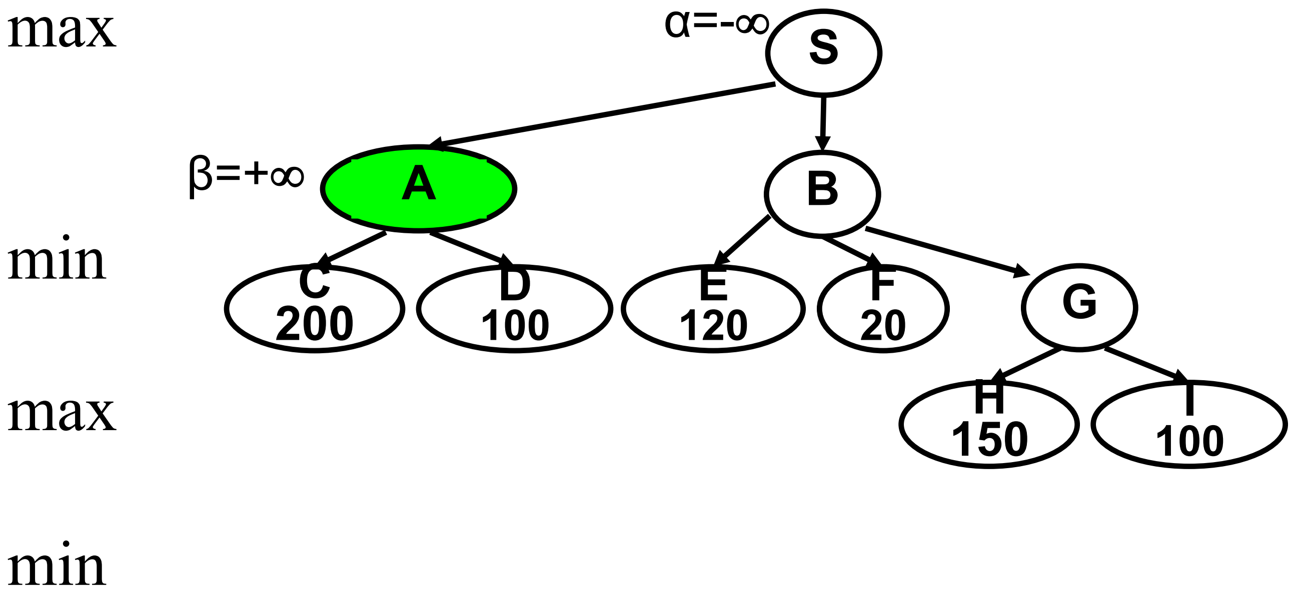
Space complexity?

- $O(bm)$

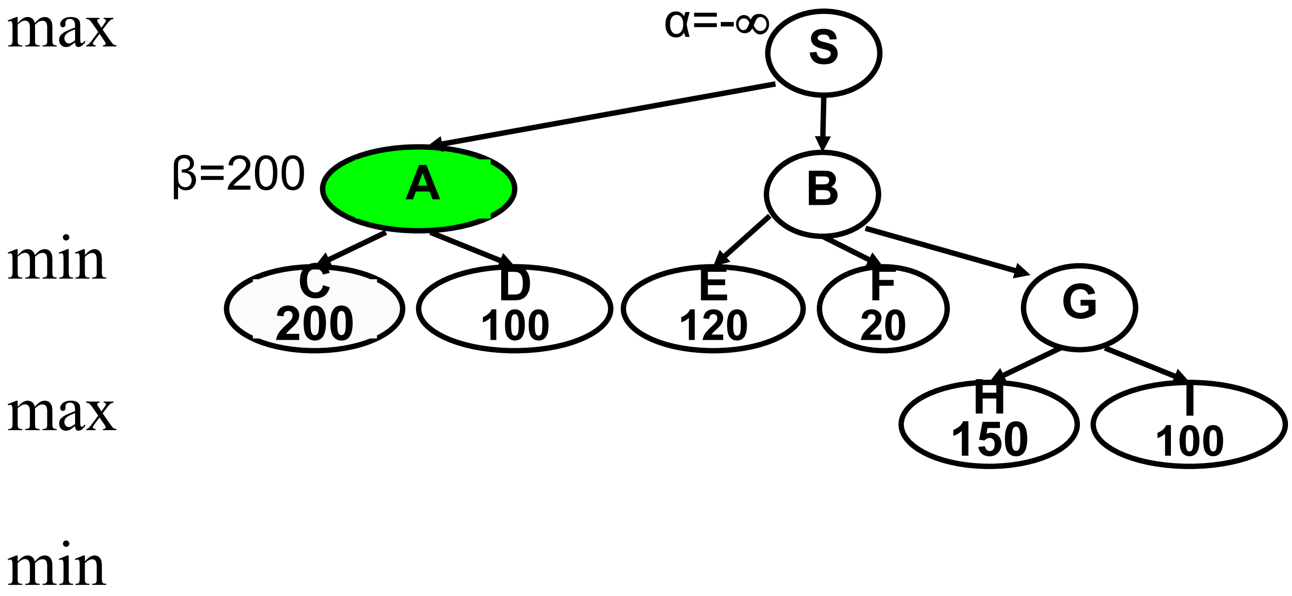
Minimax algorithm in execution



Minimax algorithm in execution

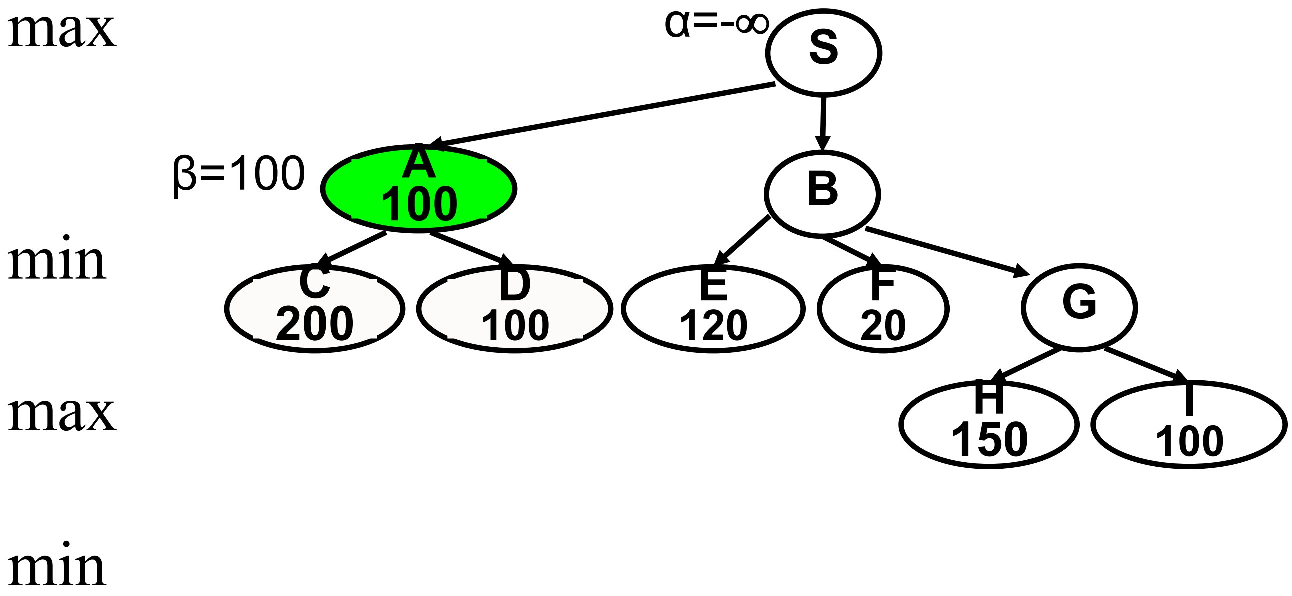


Minimax algorithm in execution

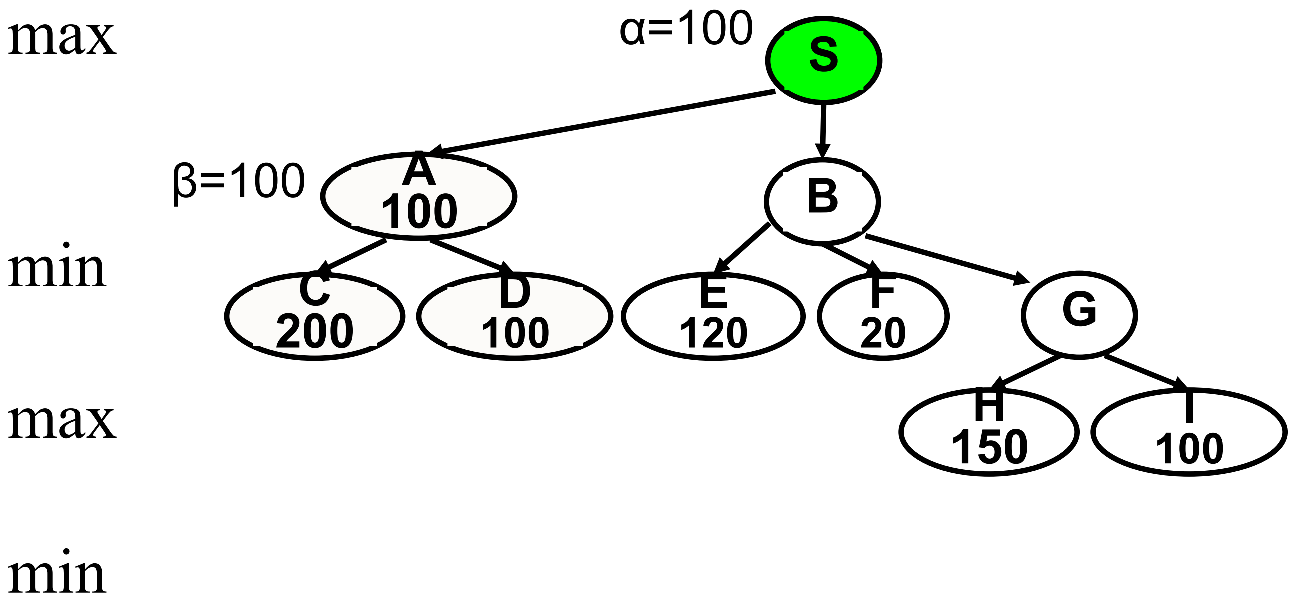


The execution on the
terminal nodes is omitted.

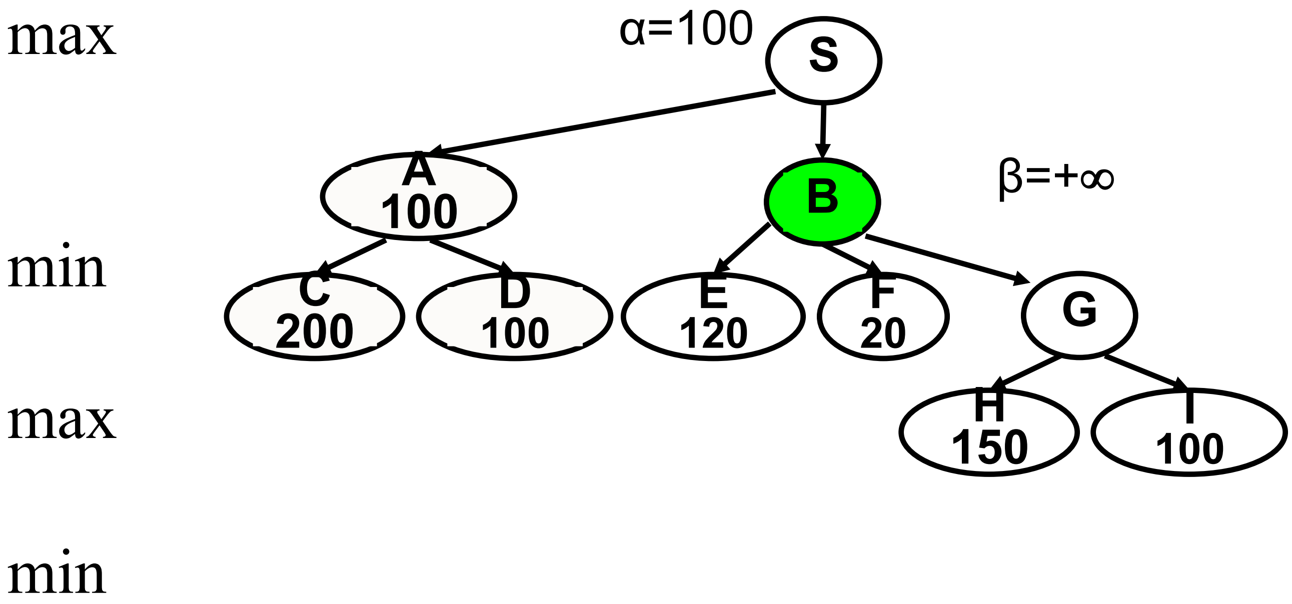
Minimax algorithm in execution



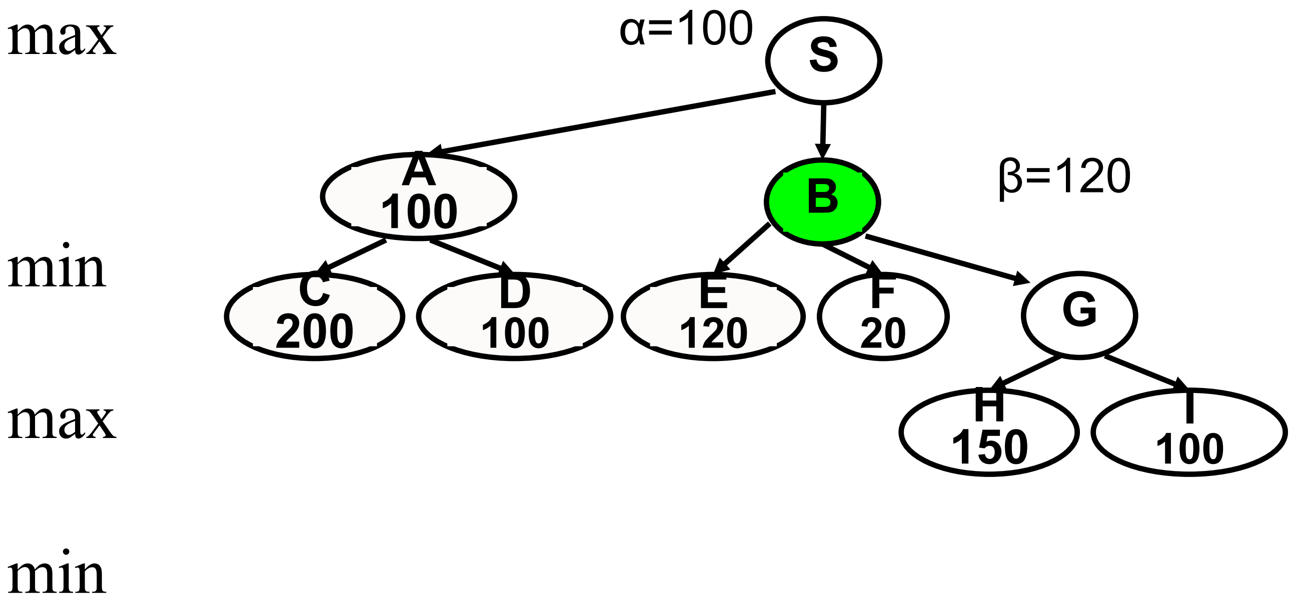
Minimax algorithm in execution



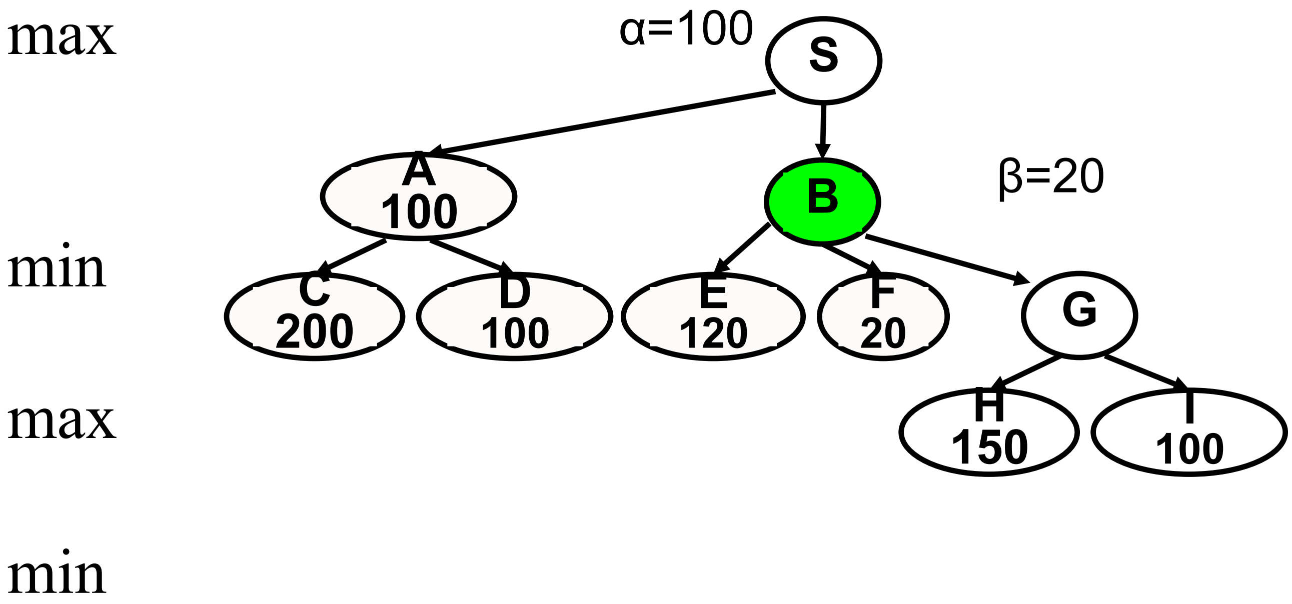
Minimax algorithm in execution



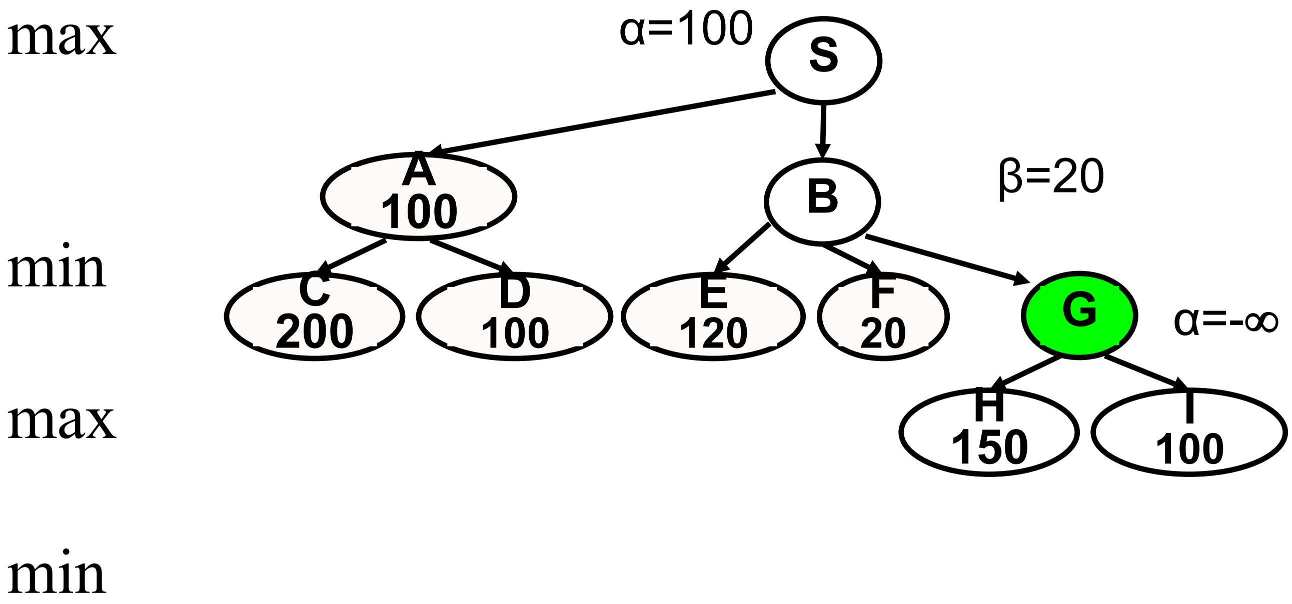
Minimax algorithm in execution



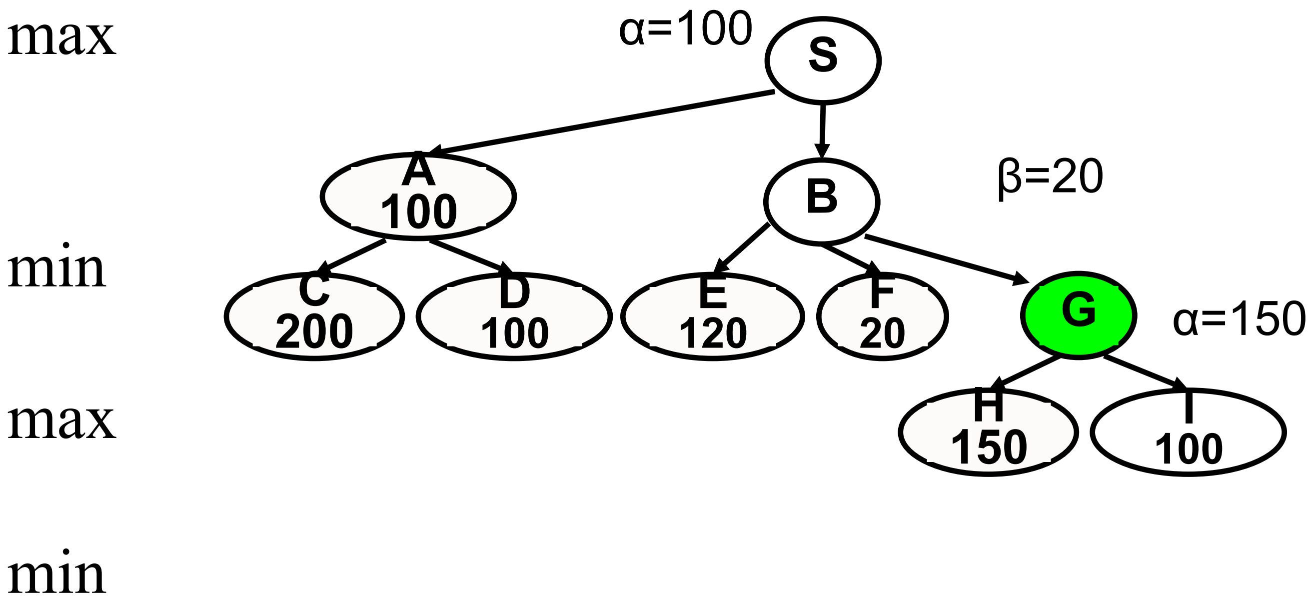
Minimax algorithm in execution



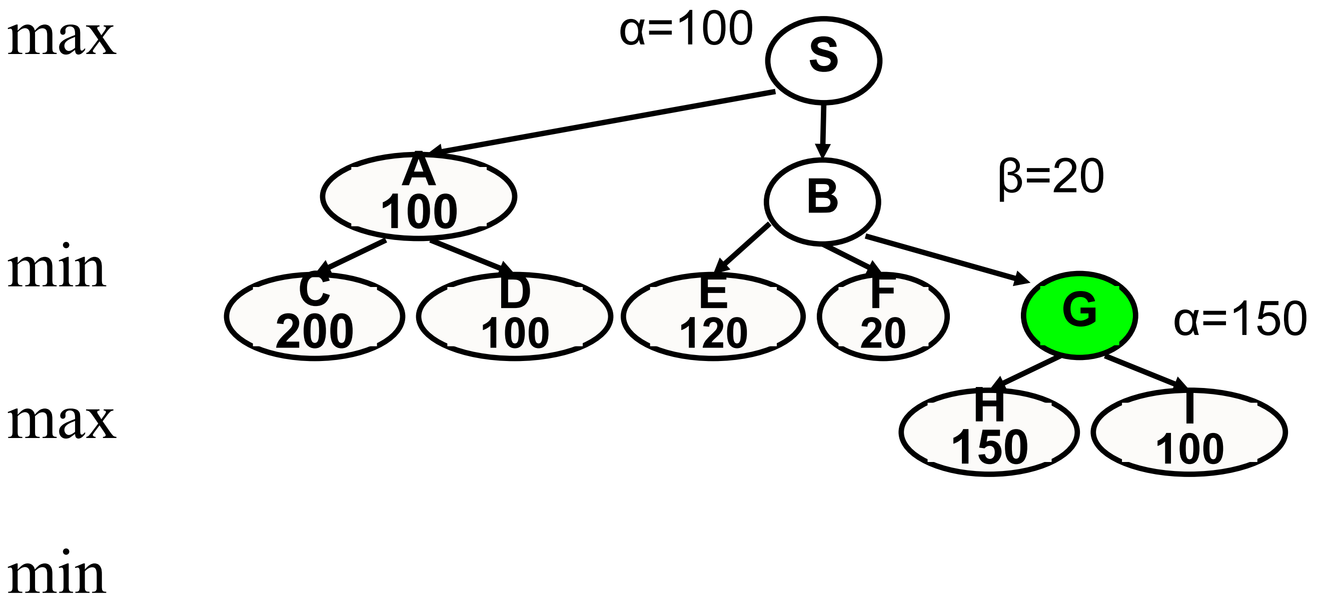
Minimax algorithm in execution



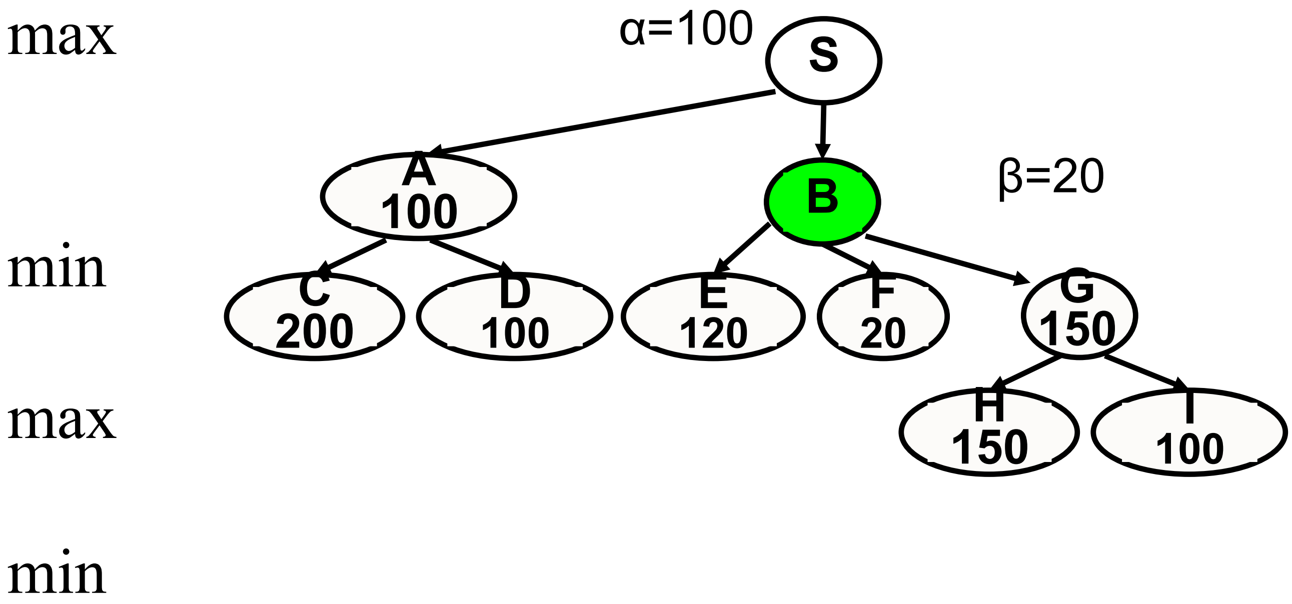
Minimax algorithm in execution



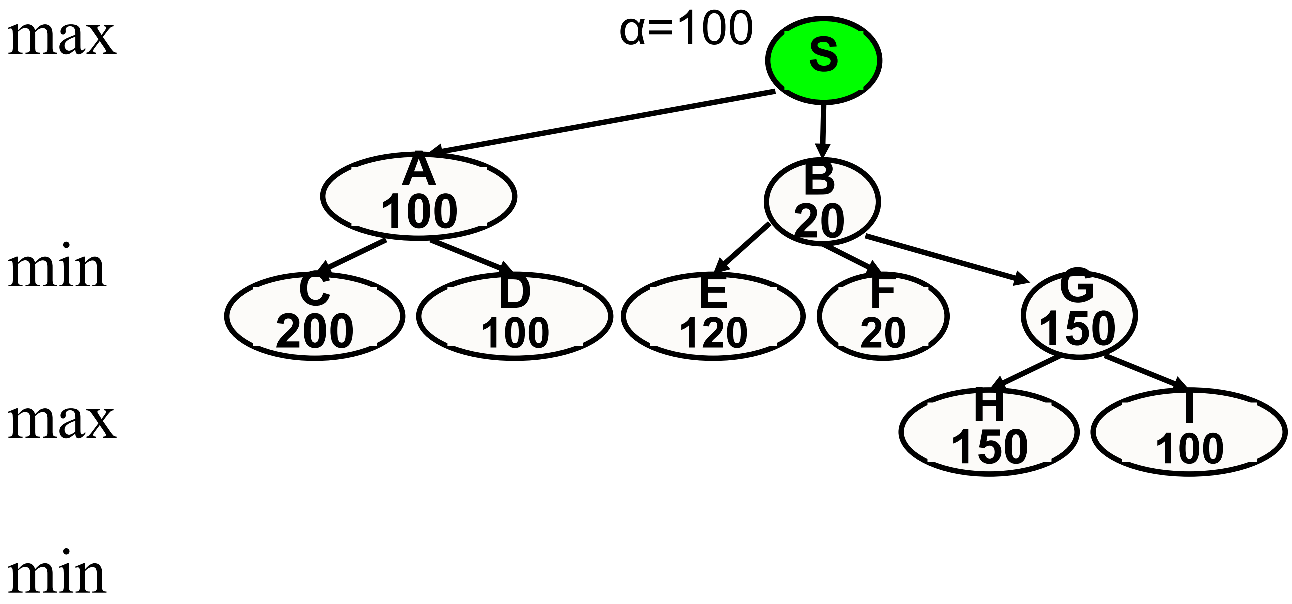
Minimax algorithm in execution



Minimax algorithm in execution



Minimax algorithm in execution



Minimax With Heuristics

Note that long games are yield 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)*

Minimax with Heuristics

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

      if ( s is a terminal state or d==0 )
      then return ( terminal/estimated value of s )
      else
                  α := – infinity
                  for each s' in Succ(s)
                      α := max( α , Min-value(s', d-1))
      return α
```

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

      if ( s is a terminal state or d==0)
      then return ( terminal/estimated value of s)
      else
                  β := infinity
                  for each s' in Succs(s)
                      β := min( β , Max-value(s', d-1))
      return β
```

Minimax with Heuristics

Min and Max Combined:

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

Heuristic Evaluation Functions

- e(x) often 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)

Summary

- Intro to game theory
  - Characterize games by various properties
- Sequential games
  - Game trees, game-theoretic/minimax value, minimax algo
- Improving our search
  - Using heuristics

Acknowledgements: Developed from materials by Yingyu Liang (University of Wisconsin), inspired by Haifeng Xu (UVA).