CS 540 Introduction to Artificial Intelligence
Game II

Yingyu Liang
University of Wisconsin-Madison

Nov 30, 2021

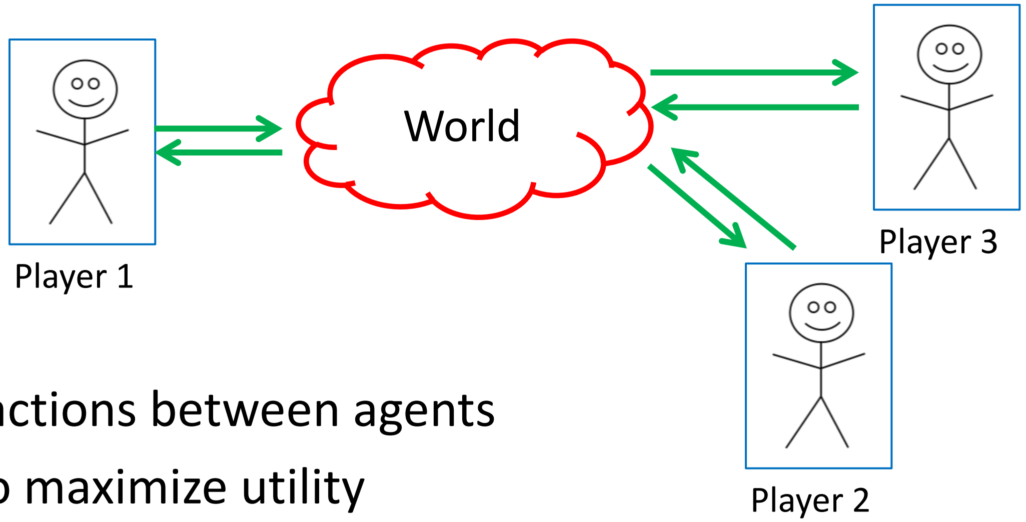
Based on slides by Fred Sala

Outline

- Review of game theory basics
  - Properties, sequential games
- Speeding up sequential game search
  - Heuristics, pruning, random search
- Simultaneous Games
  - Normal form, strategies, dominance, Nash equilibrium

Review of Games: Multiple Agents

Games setup: **multiple** agents



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

Review of Games: Properties

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

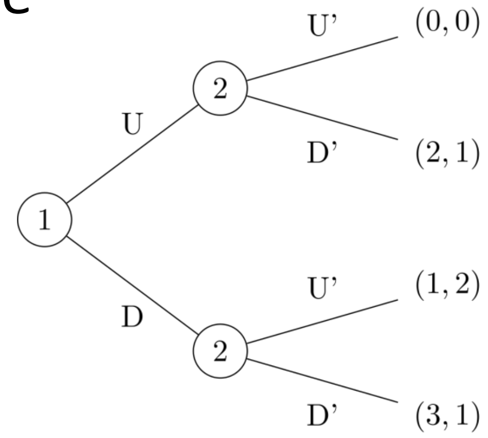
- **Number** of agents/players
- State & action spaces: **discrete** or **continuous**
- **Finite** or **infinite**
- **Deterministic** or **random**
- **Sum**: zero or positive or negative
- **Sequential** or **simultaneous**

Wiki

Sequential Games

Games with multiple moves

- Represent with a **tree**
- Find strategies: perform search over the tree



Wiki

II-Nim: Example Sequential Game

2 piles of sticks, each with 2 sticks.

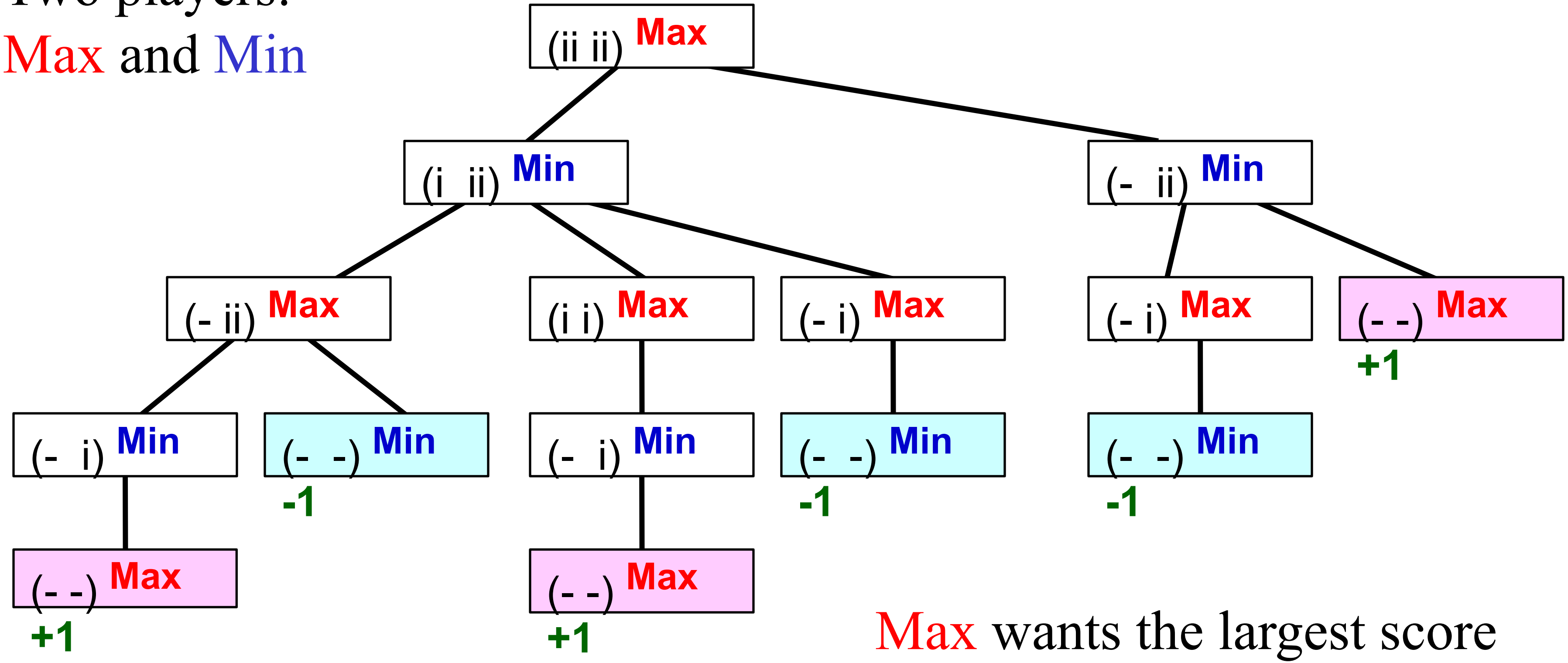
- Each player takes one or more sticks from pile
- Take last stick: lose

(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 tree for II-Nim

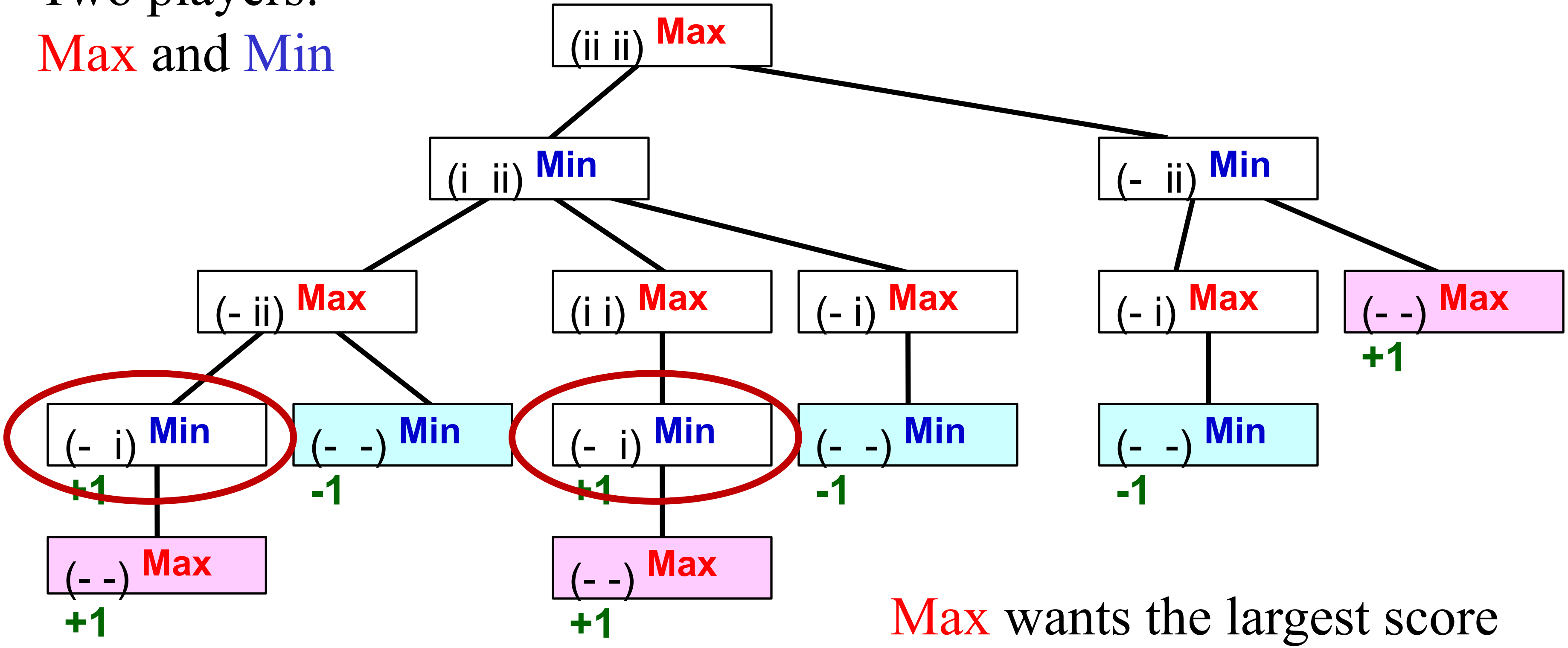
Two players:
Max and Min



Max wants the largest score
Min wants the smallest score

Game tree for II-Nim

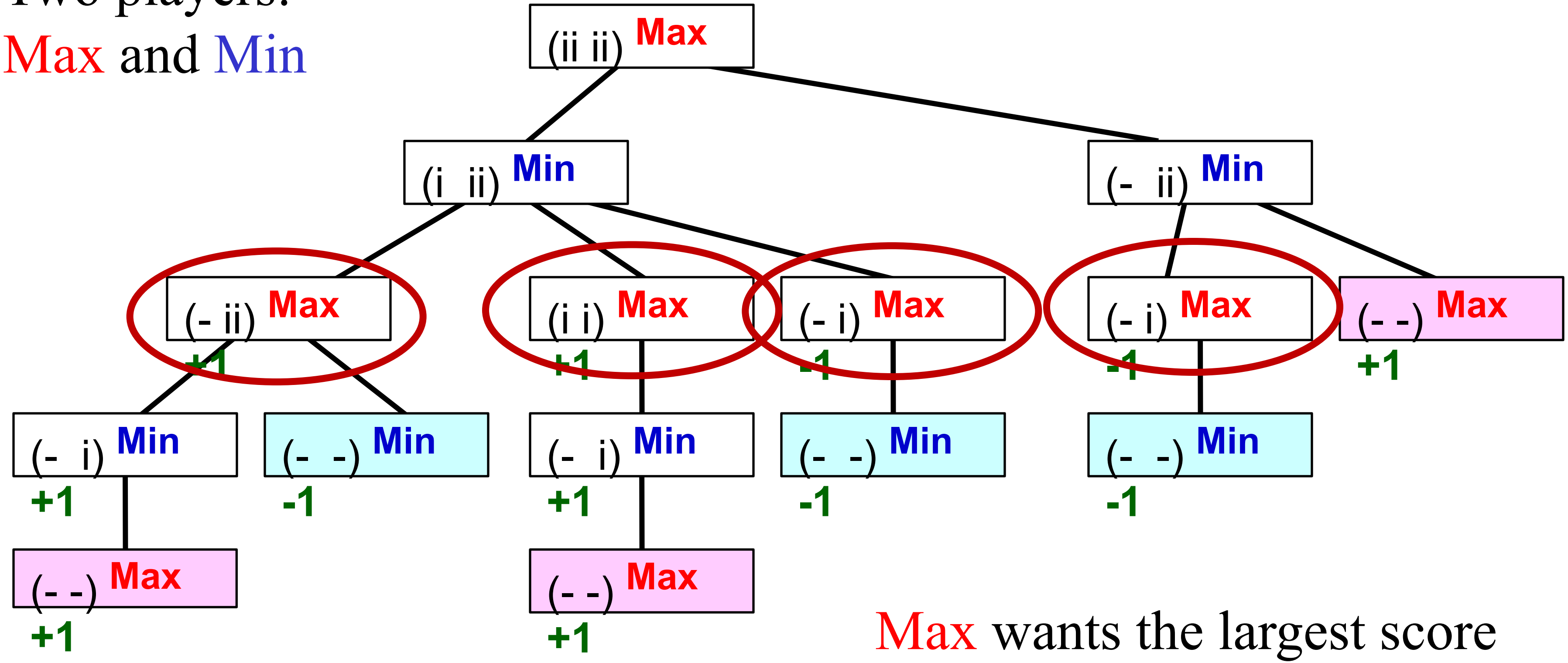
Two players:
Max and Min



Max wants the largest score
Min wants the smallest score

Game tree for II-Nim

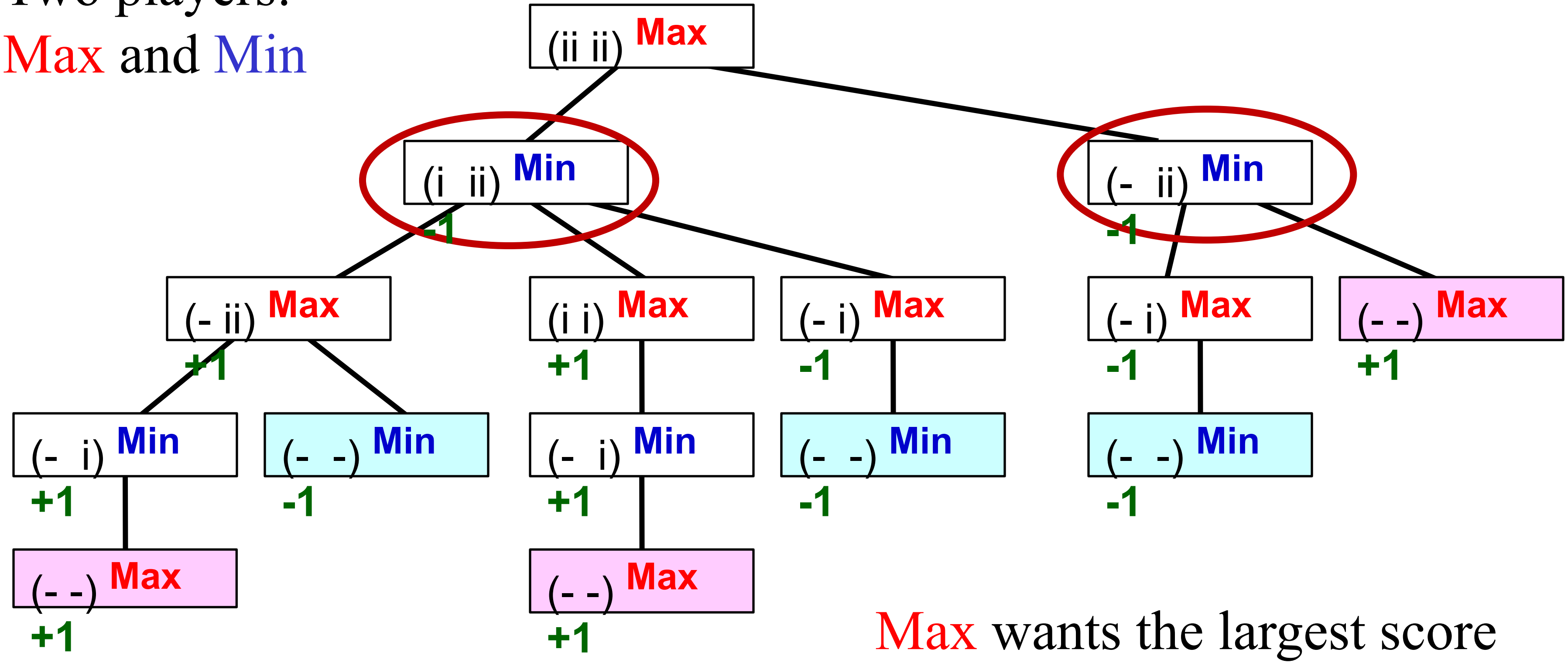
Two players:
Max and Min



Max wants the largest score
Min wants the smallest score

Game tree for II-Nim

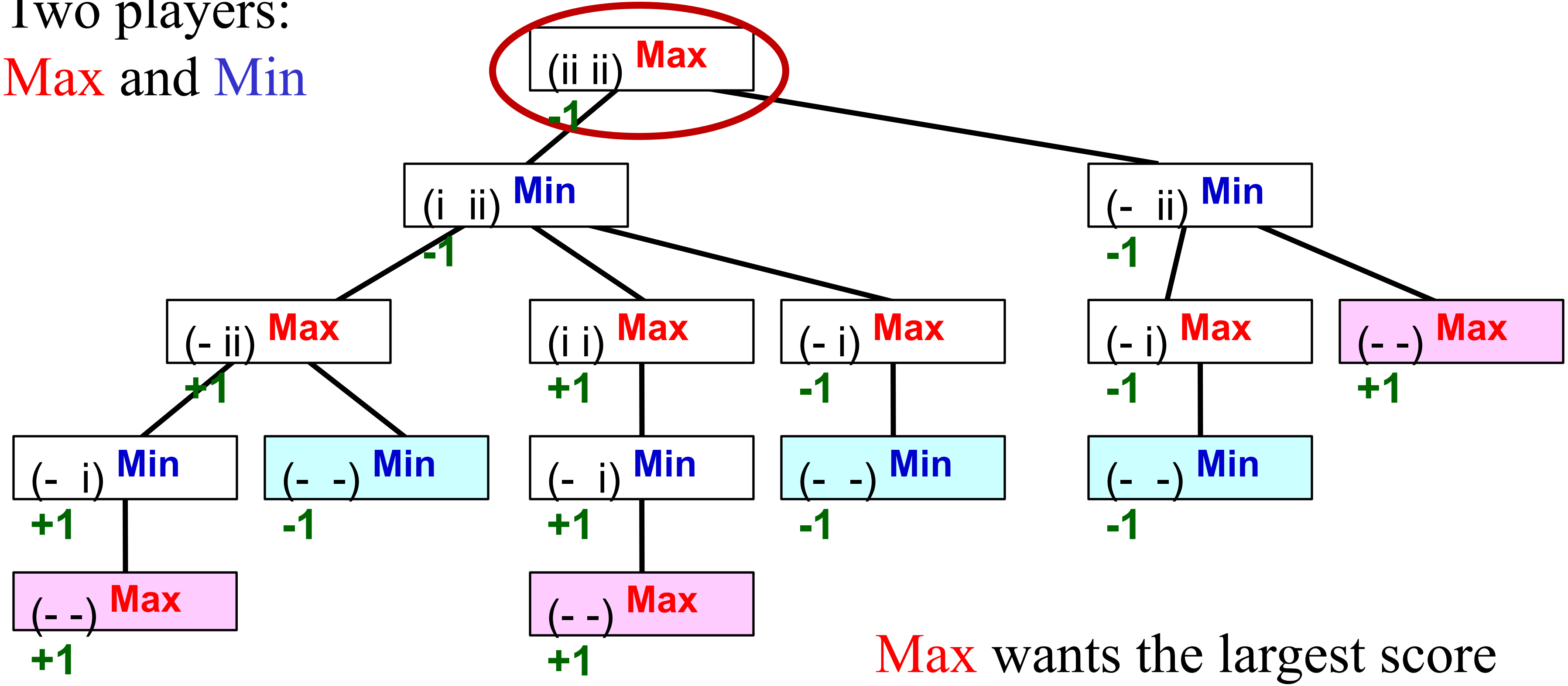
Two players:
Max and Min



Max wants the largest score
Min wants the smallest score

Game tree for II-Nim

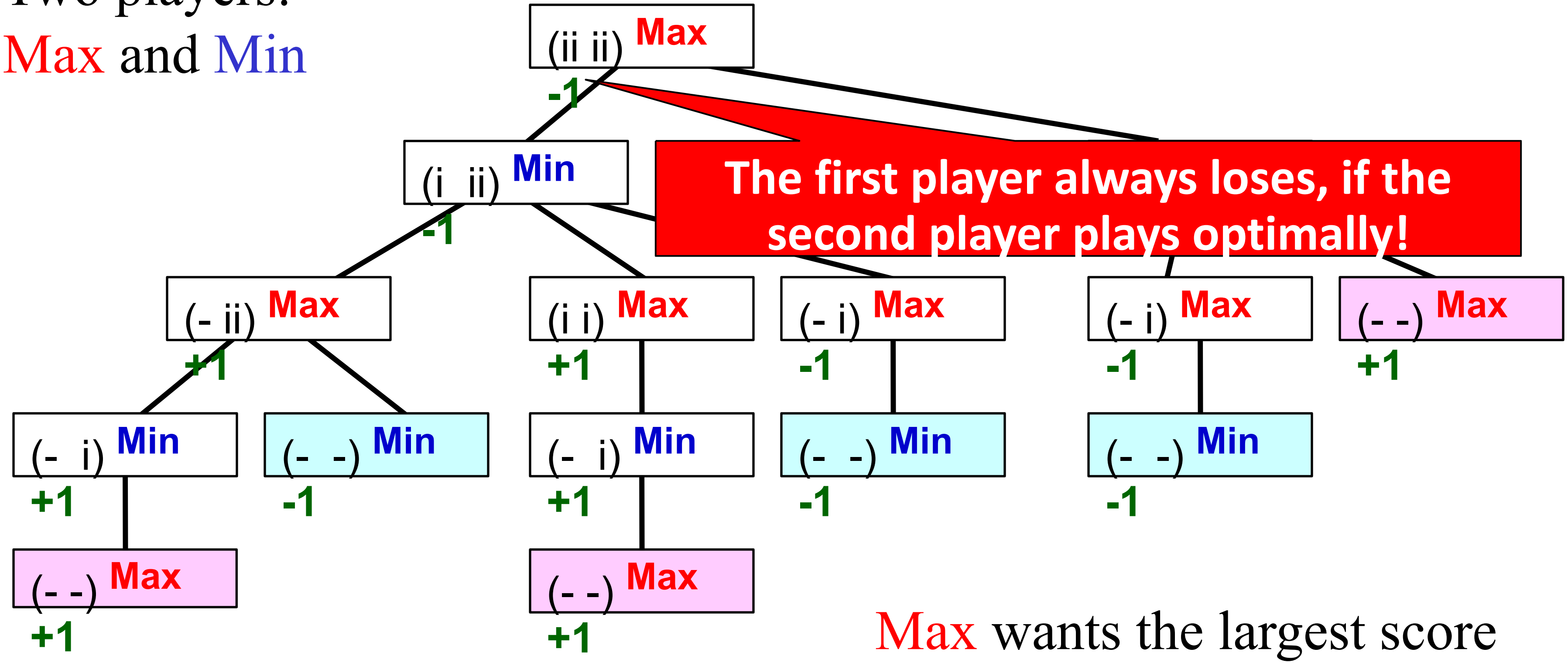
Two players:
Max and Min



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 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
```

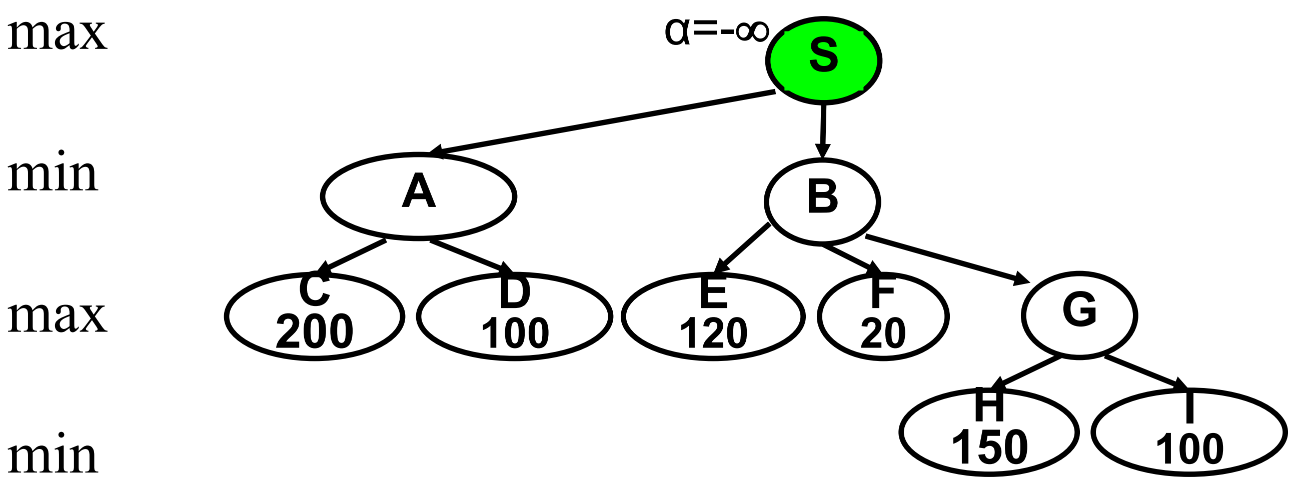
Time complexity?

- $O(b^m)$

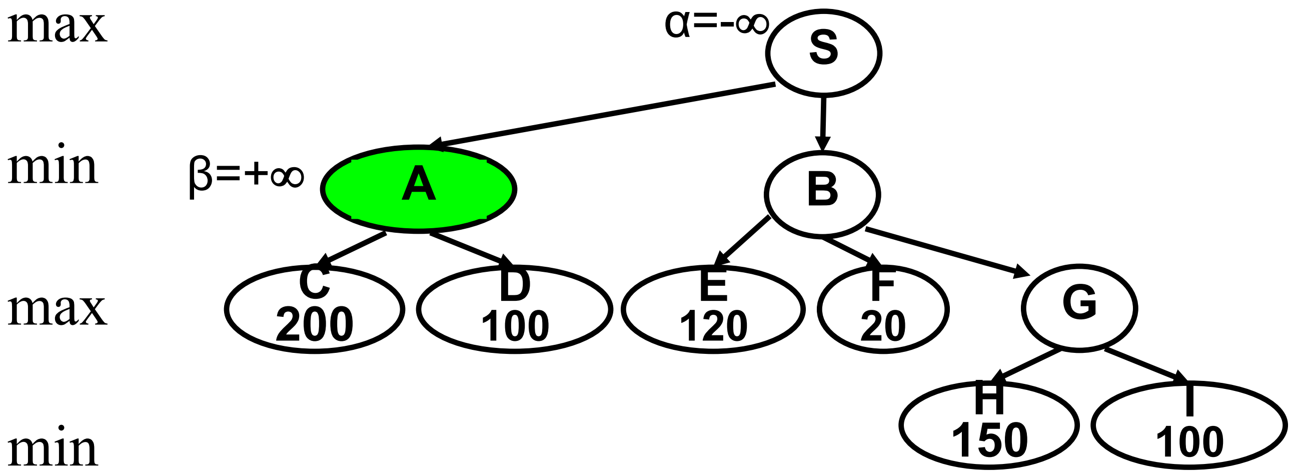
Space complexity?

- $O(bm)$

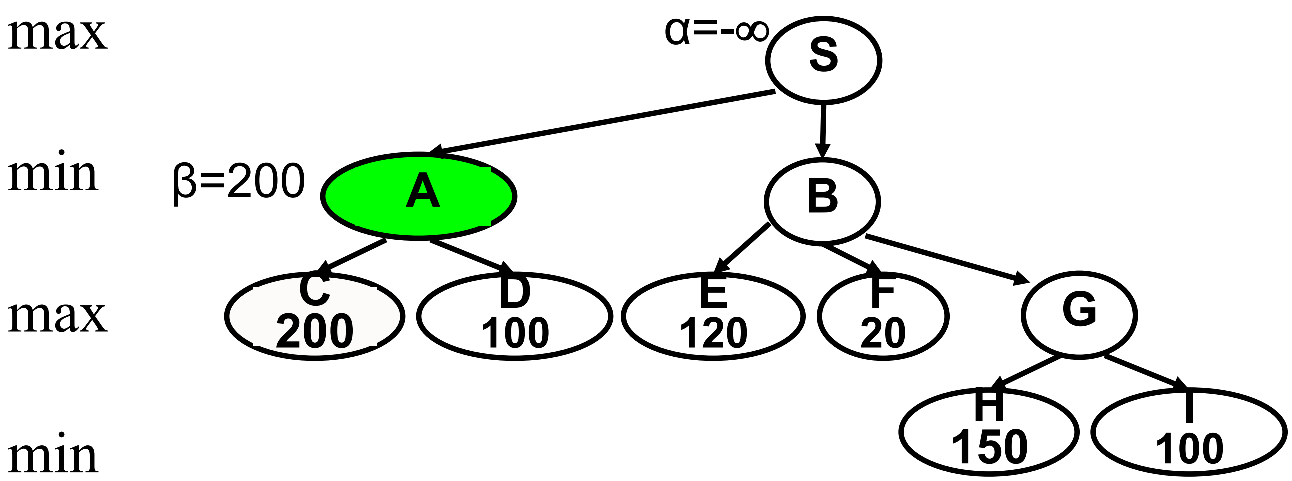
Minimax algorithm in execution



Minimax algorithm in execution

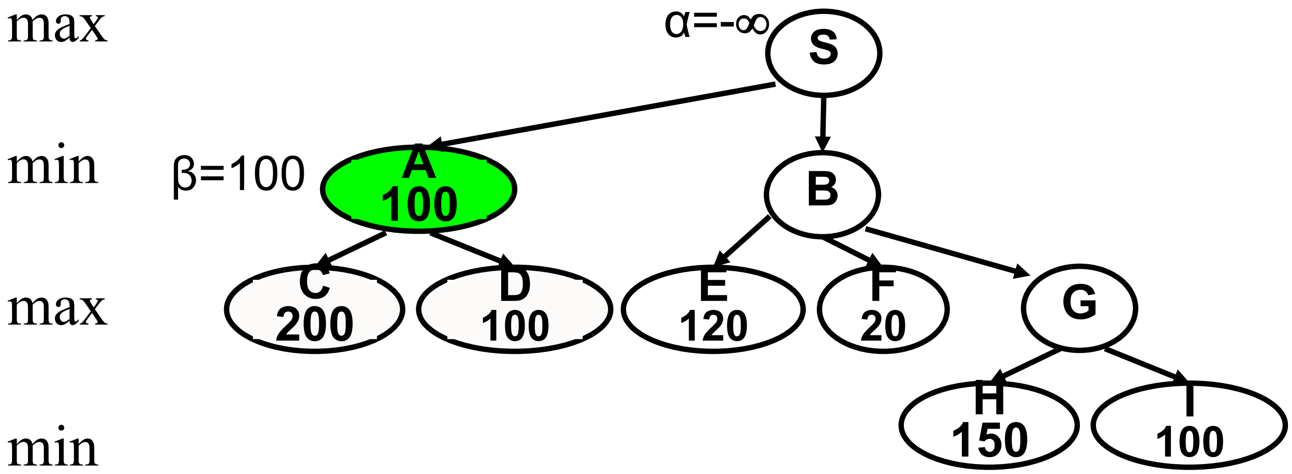


Minimax algorithm in execution

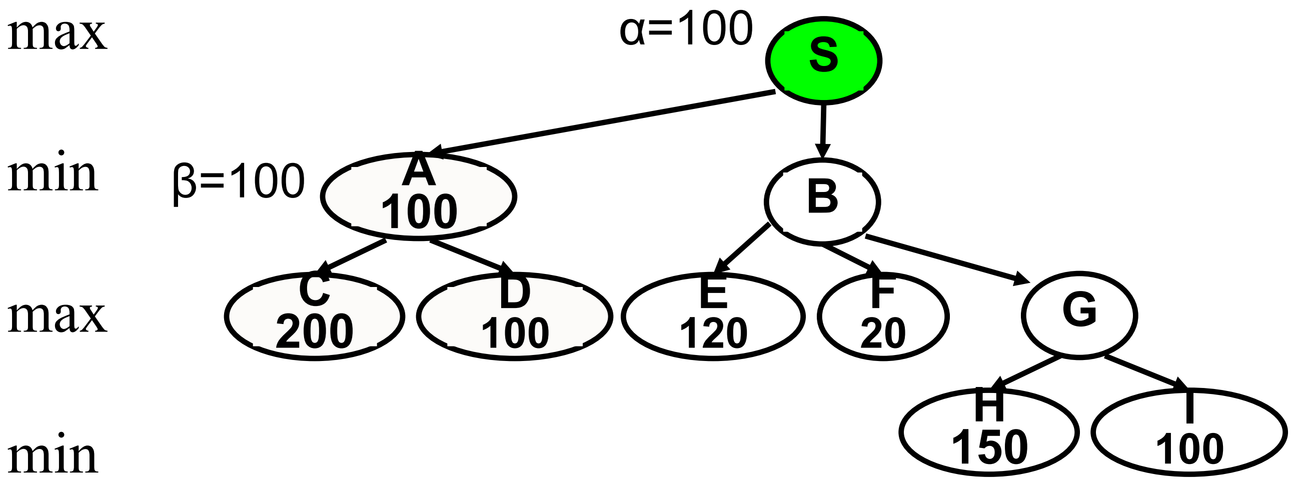


The execution on the terminal nodes is omitted.

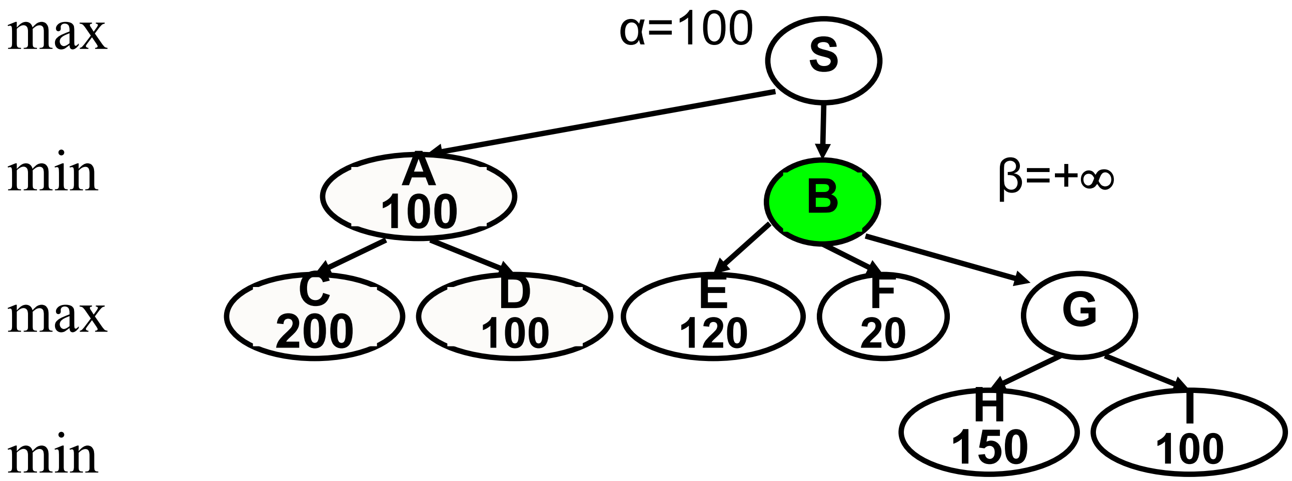
Minimax algorithm in execution



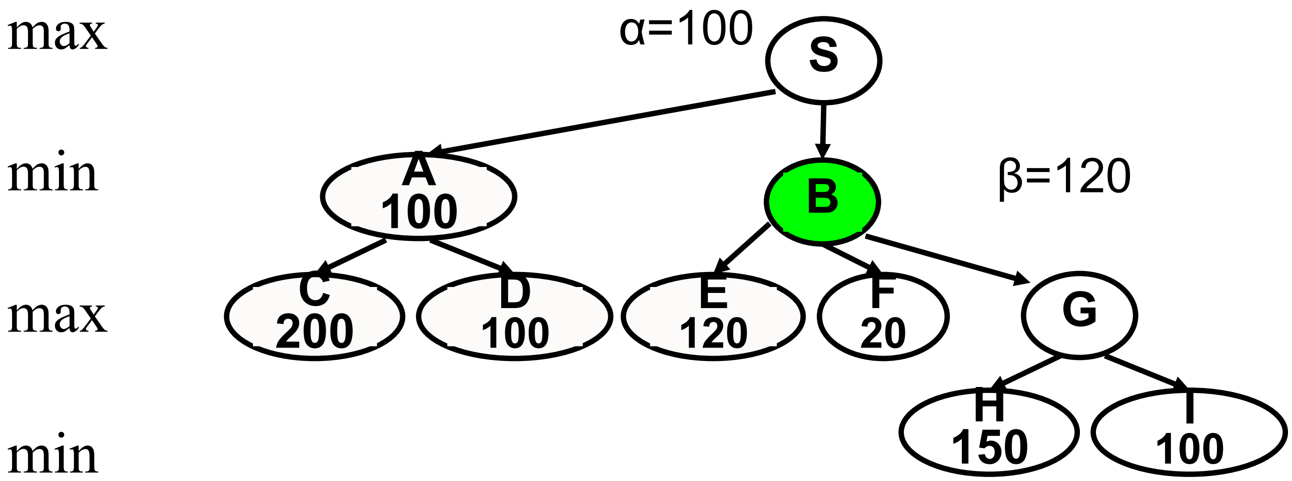
Minimax algorithm in execution



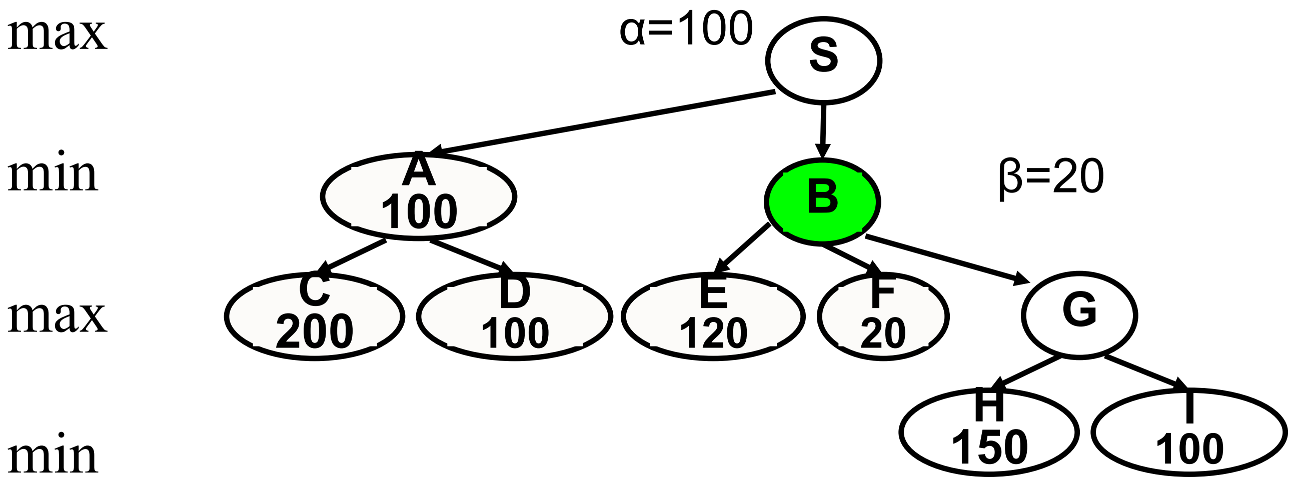
Minimax algorithm in execution



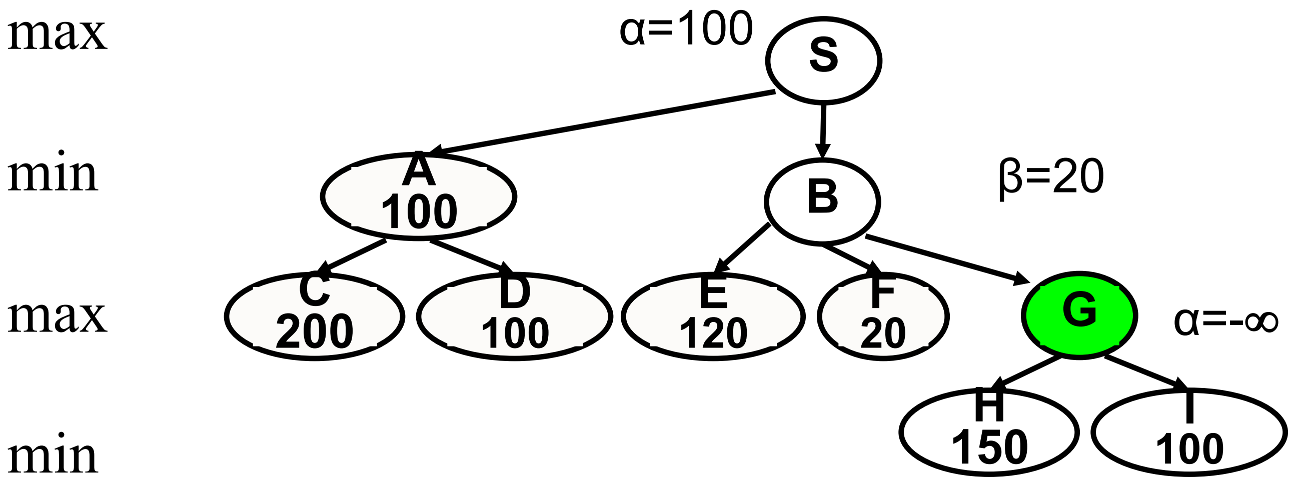
Minimax algorithm in execution



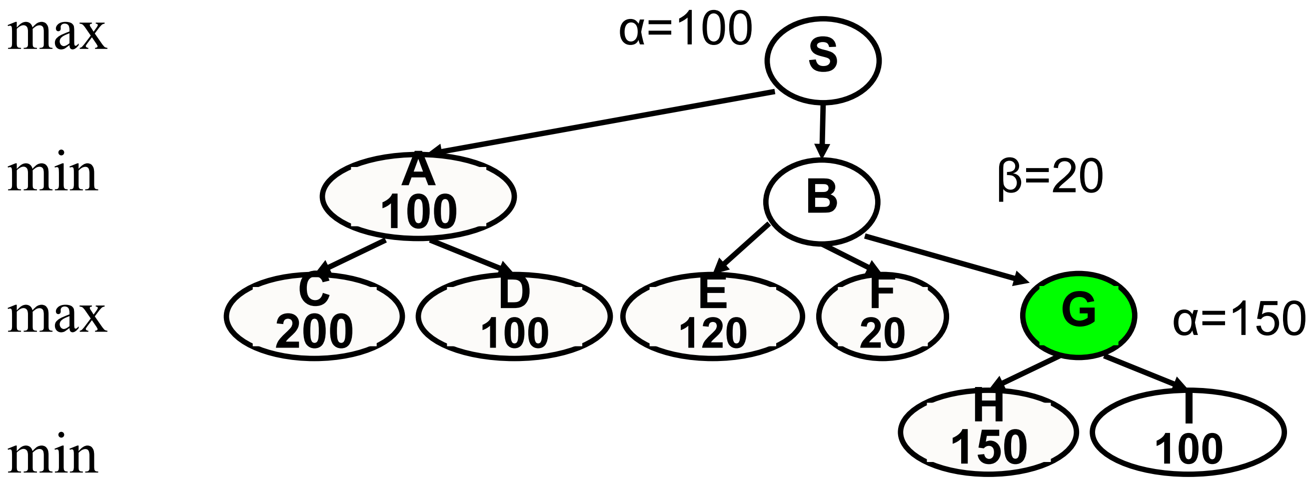
Minimax algorithm in execution



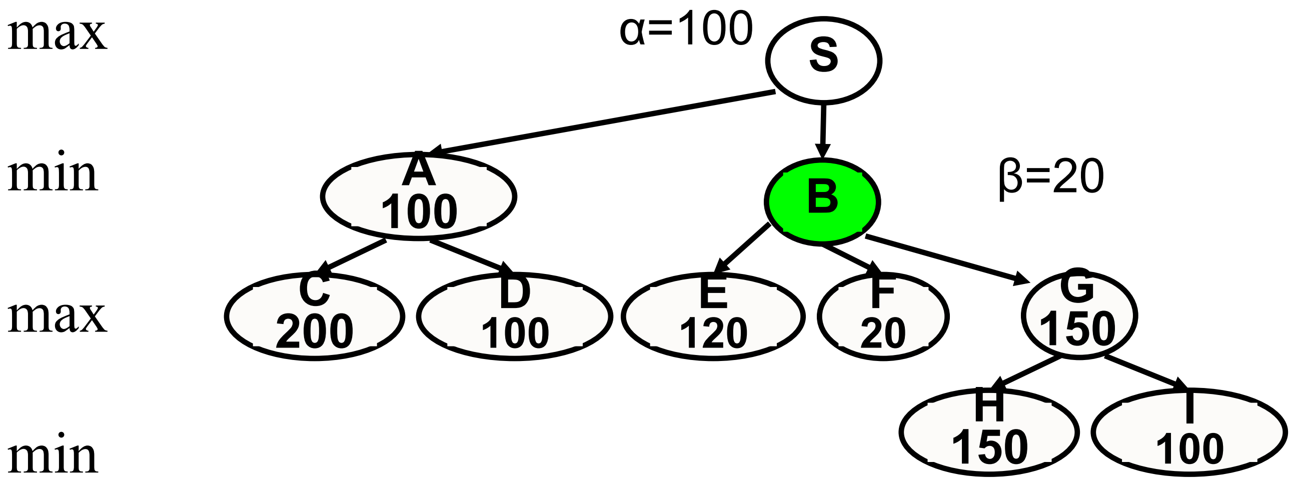
Minimax algorithm in execution



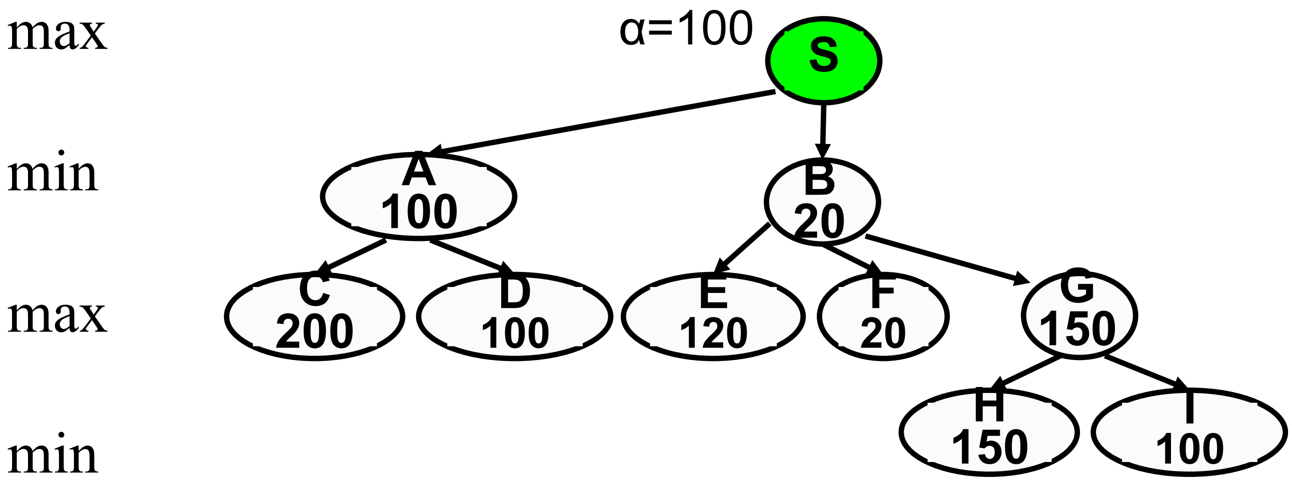
Minimax algorithm in execution



Minimax algorithm in execution



Minimax algorithm in execution



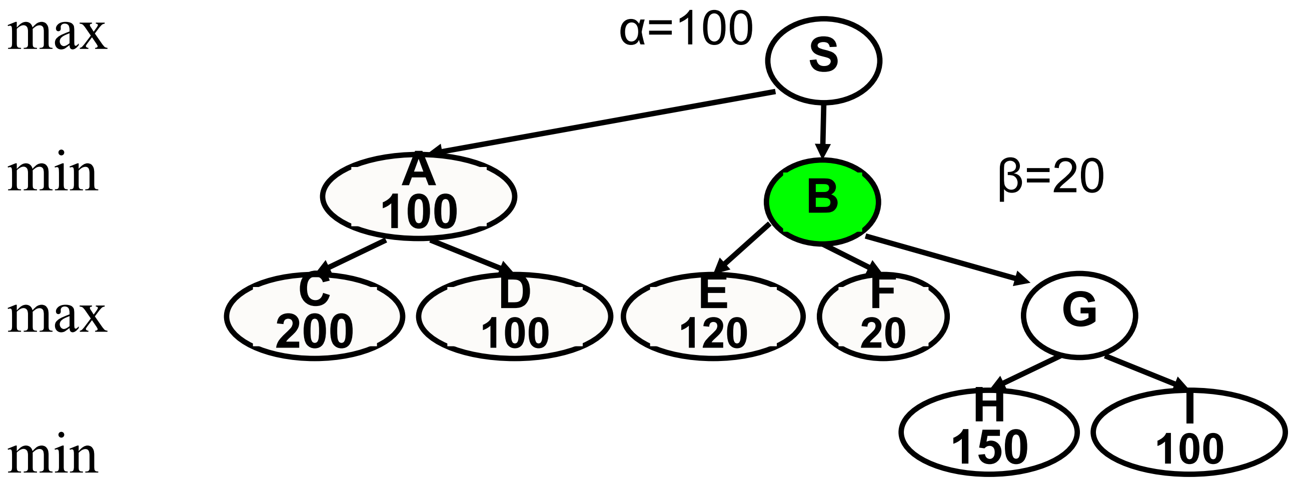
Can We Do Better?

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: when we are sure that pruning them doesn't affect the minimax value

Minimax algorithm in execution



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**?

- 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

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)*

```
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

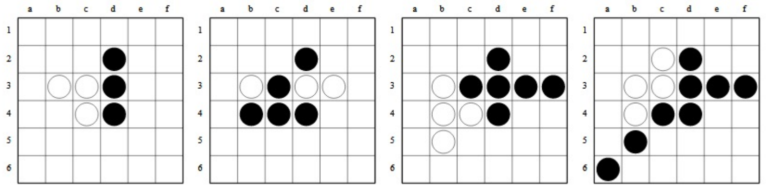
- e(x) often a weighted sum of features (like our linear models)

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

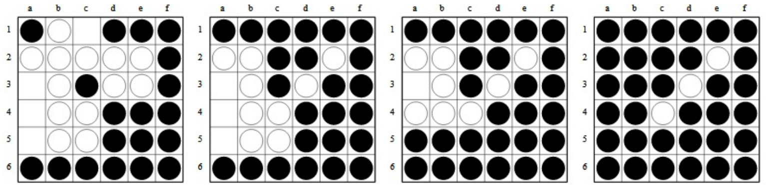
- Chess example: $f_i(x)$ = **difference** between number of white and black, with *i* ranging over piece types.
  - Set weights according to piece importance
  - E.g., 1(# white pawns - # black pawns) + 3(#white knights - # black knights)

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

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**

Simultaneous Games

The players make moves simultaneously

- Can express reward with a simple diagram
- Ex: for prisoner's dilemma

Player 2 / Player 1	*Stay silent*	*Betray*
Stay silent	−1, −1	−3, 0
Betray	0, −3	−2, −2

Normal Form

Mathematical description of simult. games. Has:

- n players $\{1,2,\ldots,n\}$
- Player i strategy a_i from A_i. **All**: $a = (a_1, a_2, \ldots, a_n)$
- Player i gets rewards $u_i(a)$ for any outcome
  - **Note**: reward depends on other players!

- Setting: all of these spaces, rewards are **known**

Example of Normal Form

Ex: Prisoner's Dilemma

Player 2 / Player 1	*Stay silent*	*Betray*
Stay silent	−1, −1	−3, 0
Betray	0, −3	−2, −2

- 2 players, 2 actions: yields 2x2 matrix
- Strategies: {Stay silent, betray} (i.e, binary)
- Rewards: {0,-1,-2,-3}

Dominant Strategies

Let’s analyze such games. Some strategies are better

- Dominant strategy: if a_i better than a_i' *regardless* of what other players do, a_i is **dominant**
- I.e.,

$$u_i(a_i, a_{-i}) \geq u_i(a_i', a_{-i}) \forall a_i' \neq a_i \text{ and } \forall a_{-i}$$

All of the other entries of *a* excluding *i*

- Don’t always exist!

Dominant Strategies Example

Back to Prisoner's Dilemma

- Examine all the entries: betray dominates
- Check:

Player 2 Player 1	*Stay silent*	*Betray*
Stay silent	−1, −1	−3, 0
Betray	0, −3	−2, −2

- Note: normal form helps **locate** dominant/dominated strategies.

Equilibrium

*a** is an equilibrium if all the players do not have an incentive to **unilaterally deviate**

$$u_i(a_i^*, a_{-i}^*) \geq u_i(a_i, a_{-i}^*) \quad \forall a_i \in A_i$$

- All players dominant strategies -> equilibrium
- Converse doesn't hold (don't need dominant strategies to get an equilibrium)

Pure and Mixed Strategies

So far, all our strategies are deterministic: "**pure**"

- Take a particular action, no randomness

Can also randomize actions: "**mixed**"

- Assign probabilities x_i to each action

$$x_i(a_i), \text{ where } \sum_{a_i \in A_i} x_i(a_i) = 1, x_i(a_i) \geq 0$$

- Note: have to now consider **expected rewards**

Nash Equilibrium

Consider the mixed strategy $x^* = (x_1^*, \ldots, x_n^*)$

- This is a **Nash equilibrium** if

$$u_i(x_i^*, x_{-i}^*) \geq u_i(x_i, x_{-i}^*) \quad \forall x_i \in \Delta_{A_i}, \forall i \in \{1, 2, \ldots, n\}$$

Better than doing anything else, "**best response**"

Space of probability distributions

- Intuition: nobody can **increase expected reward** by changing only their own strategy. A type of solution!

Properties of Nash Equilibrium

Major result: (Nash 1951)

- Every finite game has at least one Nash equilibrium
  - But not necessarily **pure** (i.e., deterministic strategy)
- Could be more than one!
- Searching for Nash equilibria: computationally **hard**!

Example: rock/paper/scissors has (1/3, 1/3, 1/3) as a mixed strategy NE.

Summary

- Review of game theory basics
  - Properties, sequential games
- Speeding up sequential game search
  - Heuristics, pruning, random search
- Simultaneous Games
  - Normal form, strategies, dominance, Nash equilibrium

Acknowledgements: Developed from materials by Yingyu Liang (University of Wisconsin), James Skrentny (University of Wisconsin), inspired by Haifeng Xu (UVA) and Dana Nau (University of Maryland).