CS 540 Introduction to Artificial Intelligence
Game I

Yingyu Liang
University of Wisconsin-Madison
Nov 23, 2021

Based on slides by Fred Sala

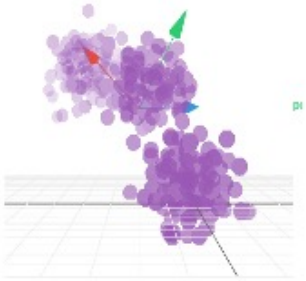
Outline

- 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

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.



Victor Powell

indoor

outdoor

Let's begin with the relation between Games and what we have learned so far.

We are going to see there are more structure in games. For example, the way we collect the data.

More General Model

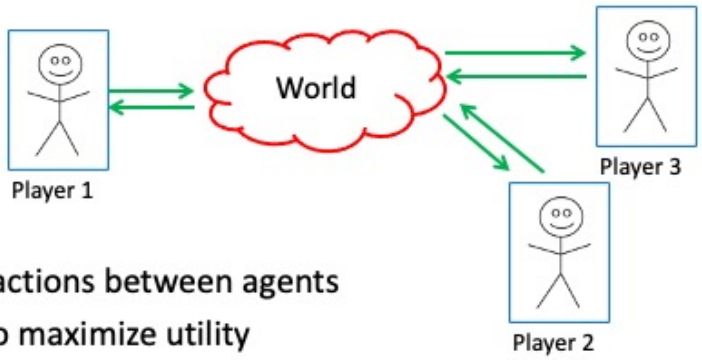
Suppose we have an **agent** **interacting** with the **world**

- 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

Reward: can be viewed as the negation of the loss. Note that different from supervised learning, here previous actions can have an impact on the world and affect the actions and observations later on. That is, the agent has some influence on the world.

Games: Multiple Agents

Games setup: **multiple** agents



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

We further model the interactions between multiple agents. We assume that each player/agent has its own reward function and wants to maximize that. However, the reward function of any player depends on all the actions of all the players and the world. So the reward function captures the interaction.

What's the different between the world and a player? Players are rational or selfish or strategic. They want to maximize their own rewards. The world is not strategic: though its state can affect the rewards but it doesn't have or want to maximize its own reward.

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

We consider a few aspects of games. Then we can divide the various kinds of games to different categories. Then we can focus on one category, build theoretical frameworks, design methods and build systems that can solve the game.

We will go over these properties one by one. Some of them are simple and self-explaining, some of them need more explanation.

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

"Finite or infinite" refers to the turns of playing the game.

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

Red \ Blue	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

The sum refers to the sum of the rewards of all the players.

In many games with two players, if one player wins, the other loses. We use zero-sum to model such games. That is, the reward of one player is the negation of the reward for the other player.

We can have other types of games with positive or negative sum of rewards. For example, prisoner's dilemma is a famous game with negative sum. We will get to that latter.

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** (rock paper scissors)
- Sequential: may or may not have **perfect information** (knowledge of all moves so far)

In sequential games, the players take turns to make a move. For example, Chess.

In simultaneous games, all players make moves at the same time, like Rock Paper Scissors. Because the players are simultaneously making moves, they do not have information about others' moves.

In sequential games, the players may have perfect information, knowing all moves of all players up to now. For example, in Chess. There are also games, when the players may not have all the information, for example, in most poker games.

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

For now, we will first consider the game theory, where we know the definition of the game, and try to build systems to solve them.

Later in reinforcement learning lectures, we are going to consider the case there are some unknown aspects of the game, and we will try to learn these unknown aspects by trying out actions and observing the rewards, so that we can solve the game.

Break & Quiz

Q 1.1: Which of these are zero-sum games?

(i) Rock, Paper, Scissors

(ii) Prisoner's Dilemma

- A. Neither
- B. (i) but not (ii)
- C. (ii) but not (i)
- D. Both

Break & Quiz

Q 1.1: Which of these are zero-sum games?

(i) Rock, Paper, Scissors

(ii) Prisoner's Dilemma

- A. Neither
- **B. (i) but not (ii)**
- C. (ii) but not (i)
- D. Both

Break & Quiz

Q 1.1: Which of these are zero-sum games?

(i) Rock, Paper, Scissors

(ii) Prisoner's Dilemma

- A. Neither (Rock, Paper, Scissors is, clearly)
- **B. (i) but not (ii)**
- C. (ii) but not (i) (Rock, Paper, Scissors is, clearly)
- D. Both (Prisoner's Dilemma is not, recall the normal form matrix)

Break & Quiz

Q 1.2: Which of these is false?

- A. Monopoly is not deterministic.
- B. A game can be sequential but not have perfect information.
- C. Chess doesn't have perfect information.
- D. Prisoner's dilemma is a simultaneous game.

Break & Quiz

Q 1.2: Which of these is false?

- A. Monopoly is not deterministic.
- B. A game can be sequential but not have perfect information.
- **C. Chess doesn't have perfect information.**
- D. Prisoner's dilemma is a simultaneous game.

Break & Quiz

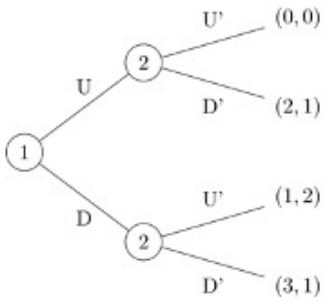
Q 1.2: Which of these is false?

- A. Monopoly is not deterministic. (True: you roll dice.)
- B. A game can be sequential but not have perfect information. (True, like poker.)
- **C. Chess doesn't have perfect information.**
- D. Prisoner's dilemma is a simultaneous game. (Also true: single round, no turns.)

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: the first player is called Max and the other 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

There are two players. They take turns to make a move. The first player makes the first move, and then the second player makes a move, and then the first player, and so on.

Since the two players' scores sum up to 0, when we describe the outcome of the game, we can just mention one player's score. The other player's score is just the negation. The convention is to use the first player's score. We just define the score of the game to be that of the first player.

So the first player wants to maximize the score of the game: that's why we call the first player Max. The second player wants to maximize its own score, that is to minimize the score of the game, so we call the second player Min.

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,-)

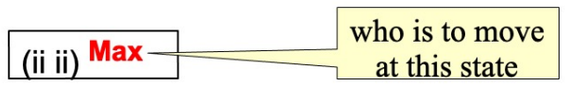
Max takes the last stick

(-,-)

Max gets score **-1**

Game tree for II-Nim

Two players:
Max and Min

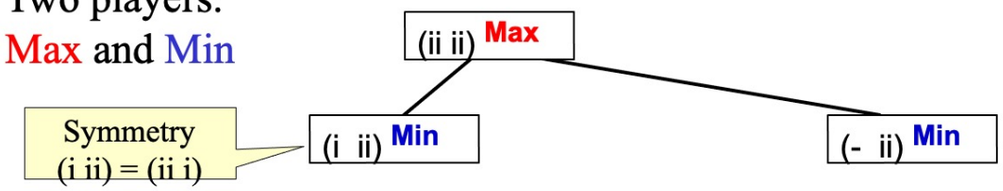


Convention: score is w.r.t. the first player Max. Min's score = – Max

Max wants the largest score
Min wants the smallest score

Game tree for II-Nim

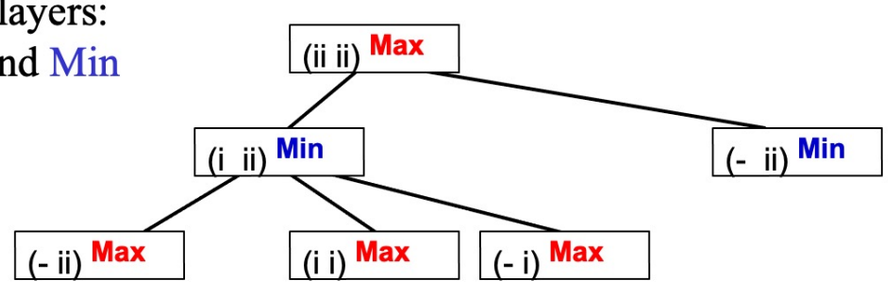
Two players:
Max and Min



Max wants the largest score
Min wants the smallest score

Game tree for II-Nim

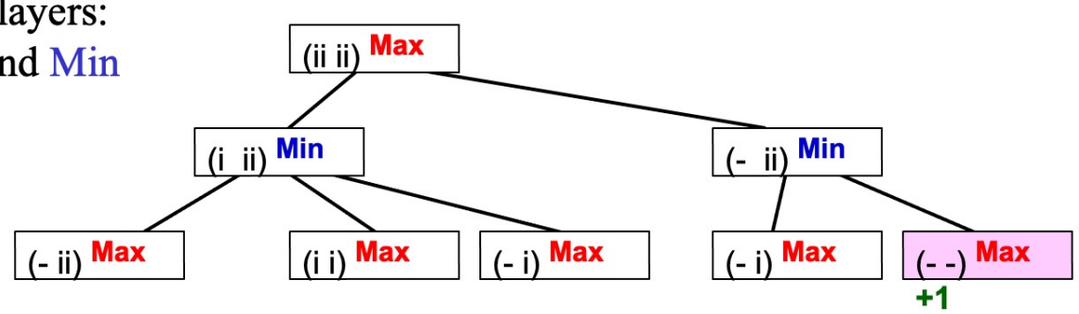
Two players:
Max and Min



Max wants the largest score
Min wants the smallest score

Game tree for II-Nim

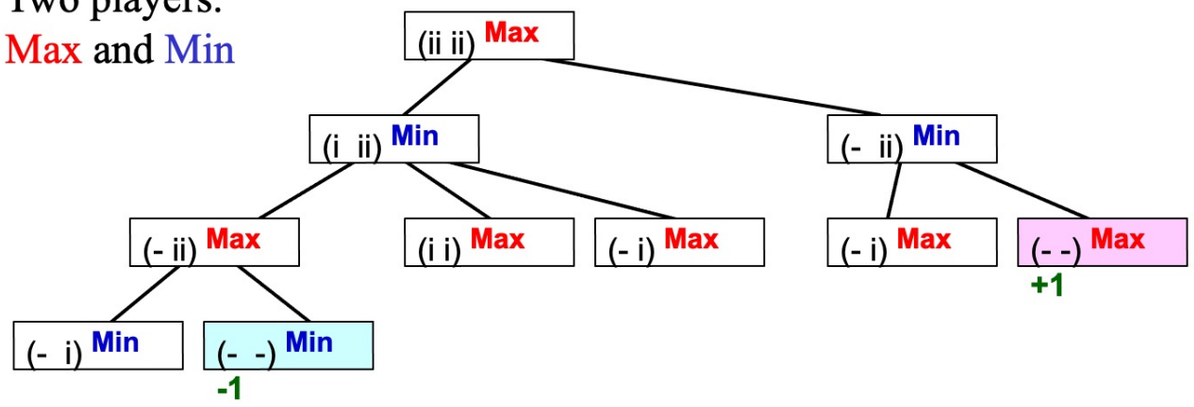
Two players:
Max and Min



Max wants the largest score
Min wants the smallest score

Game tree for II-Nim

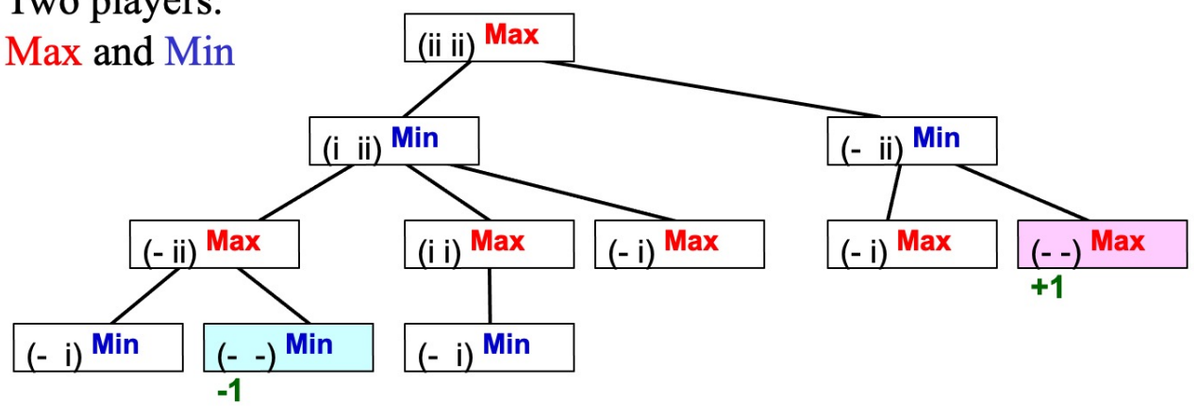
Two players:
Max and Min



Max wants the largest score
Min wants the smallest score

Game tree for II-Nim

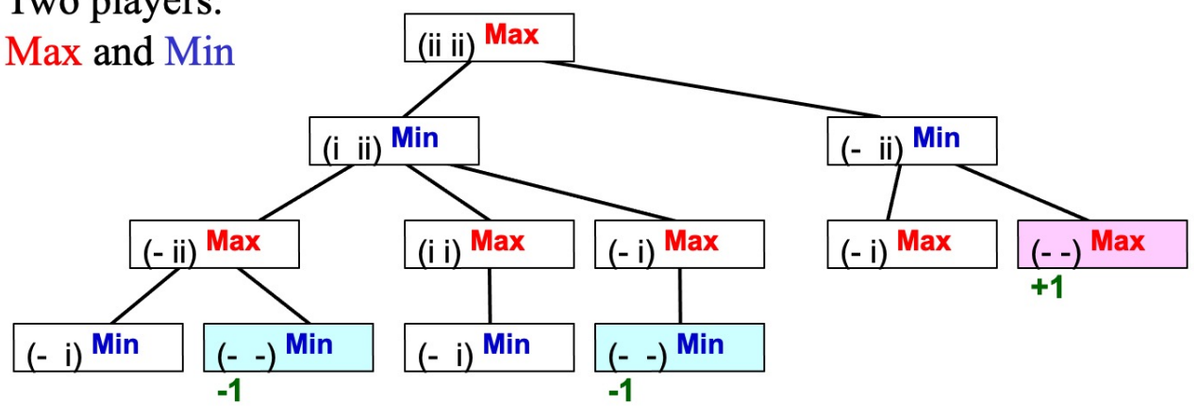
Two players:
Max and Min



Max wants the largest score
Min wants the smallest score

Game tree for II-Nim

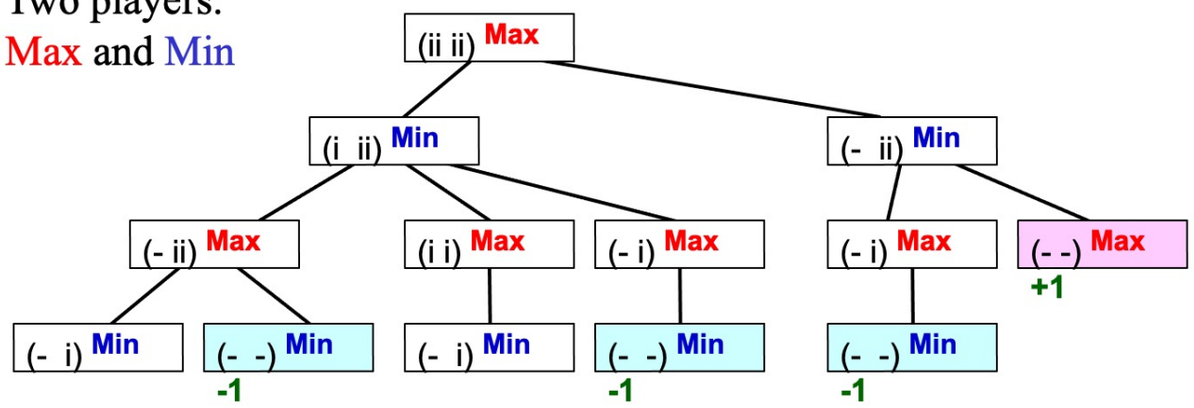
Two players:
Max and Min



Max wants the largest score
Min wants the smallest score

Game tree for II-Nim

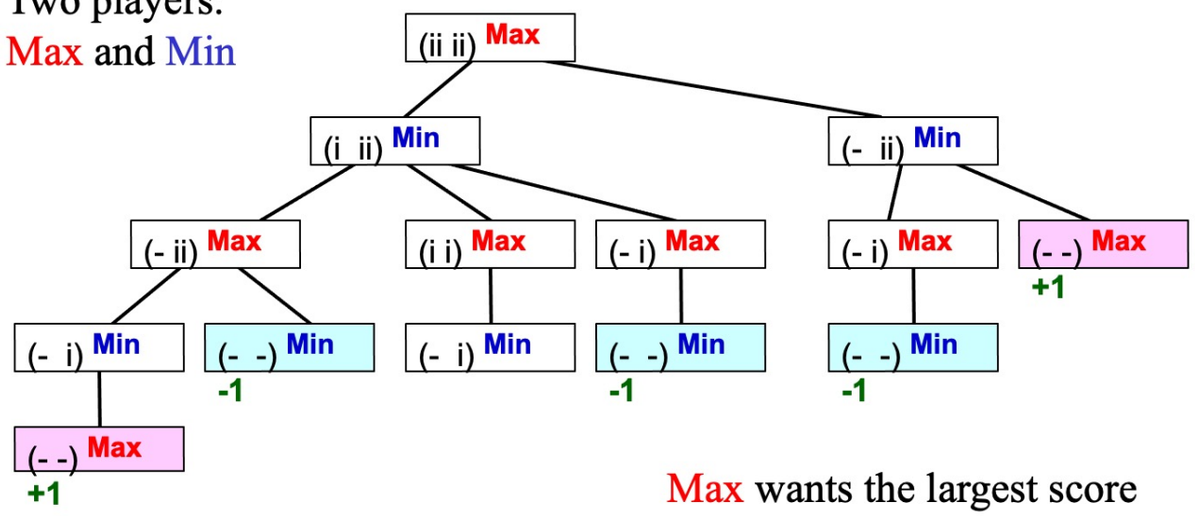
Two players:
Max and Min



Max wants the largest score
Min wants the smallest score

Game tree for II-Nim

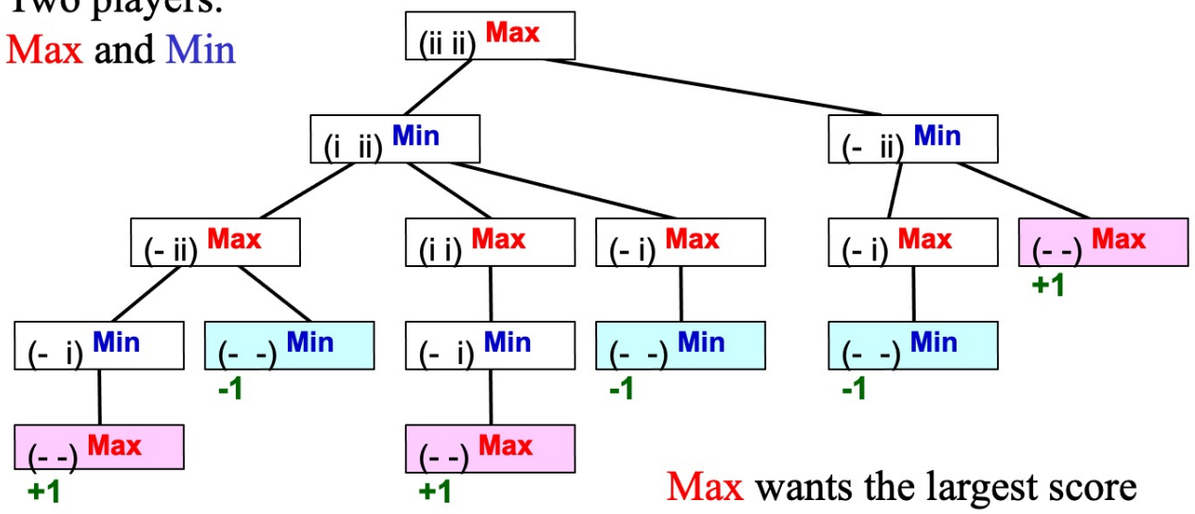
Two players:
Max and Min



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

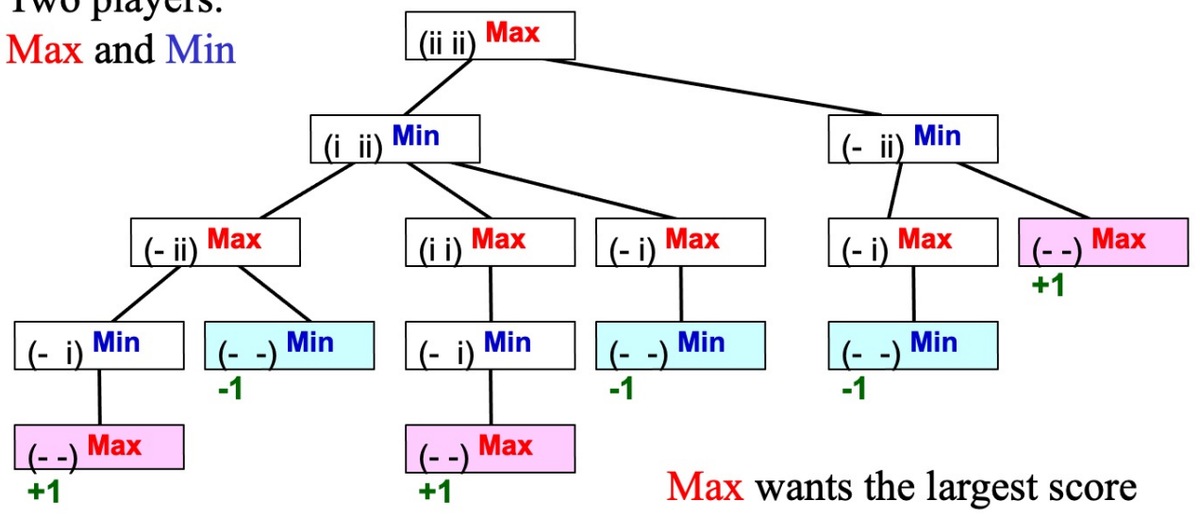
Also called **game-theoretic value**.

- 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

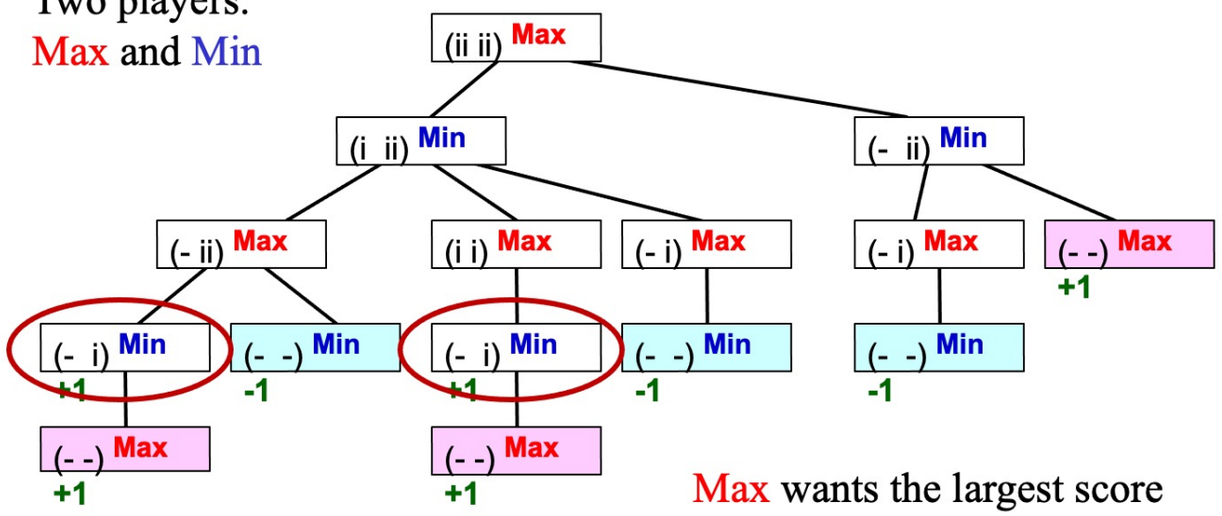
Two players:
Max and Min



Max wants the largest score
Min wants the smallest score

Game tree for II-Nim

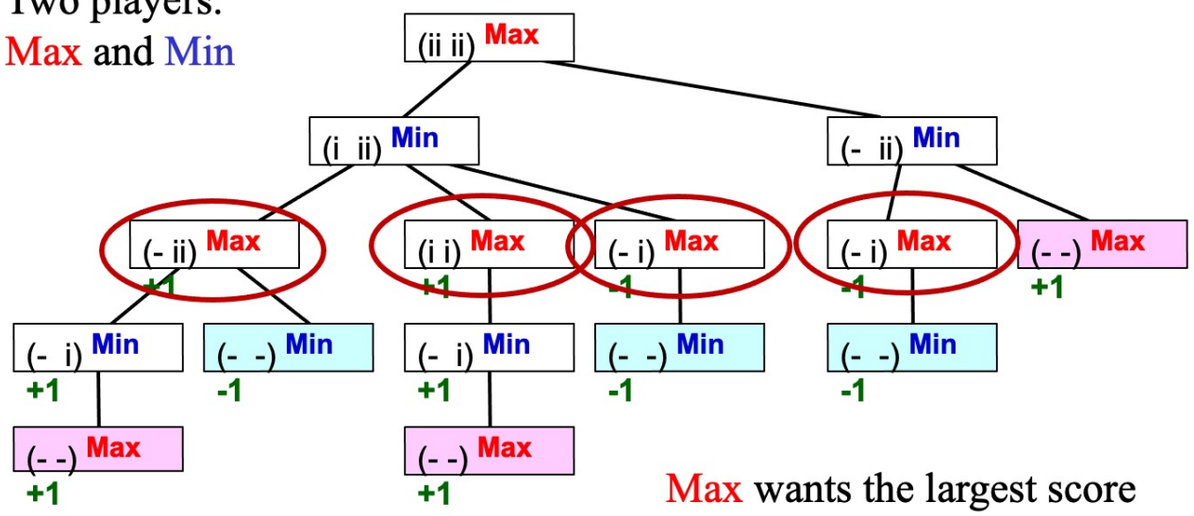
Two players:
Max and Min



Max wants the largest score
Min wants the smallest score

Game tree for II-Nim

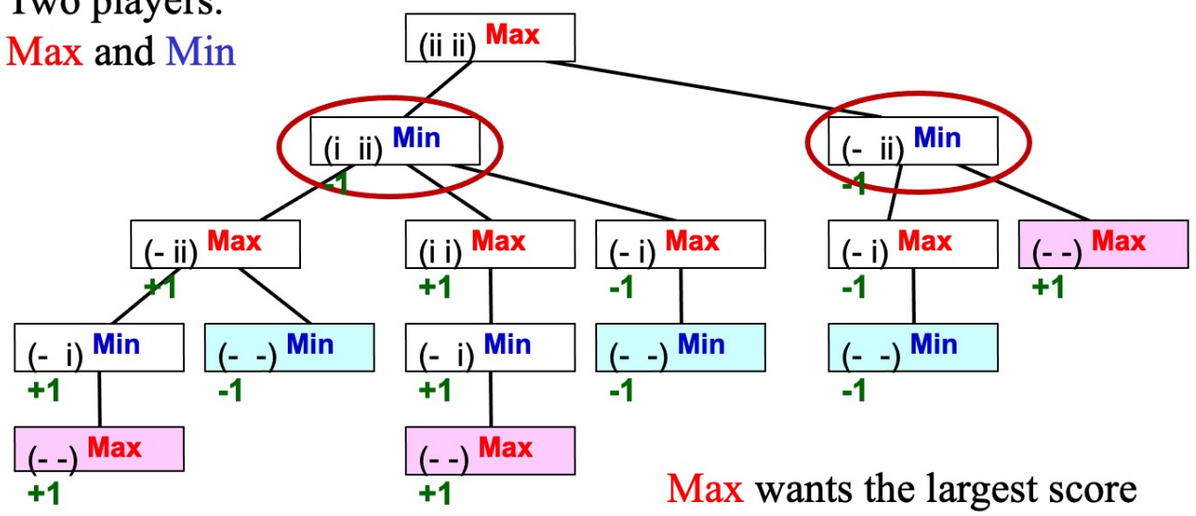
Two players:
Max and Min



Max wants the largest score
Min wants the smallest score

Game tree for II-Nim

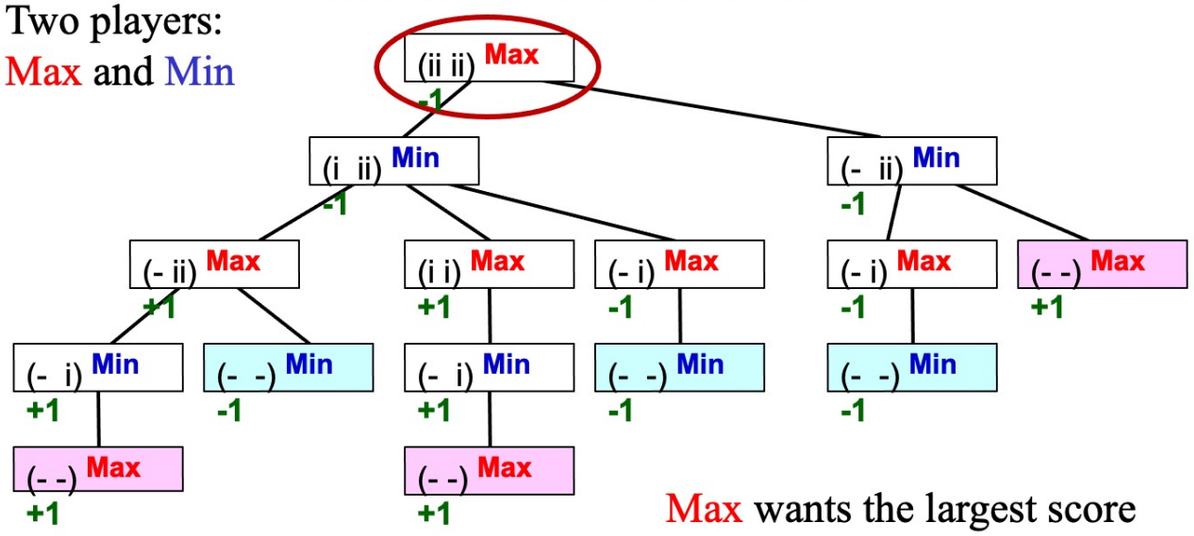
Two players:
Max and Min



Max wants the largest score
Min wants the smallest score

Game tree for II-Nim

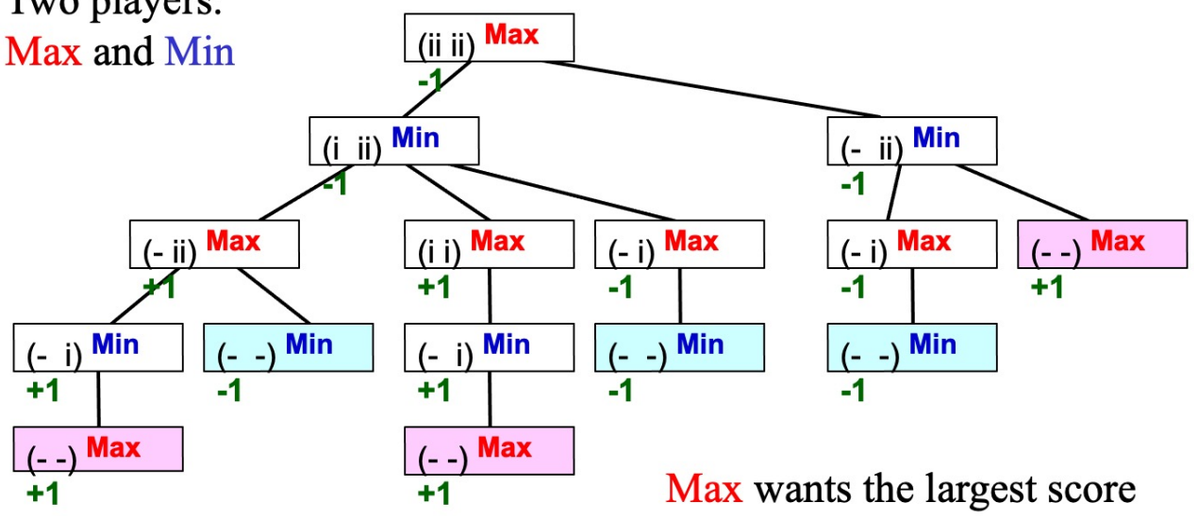
Two players:
Max and Min



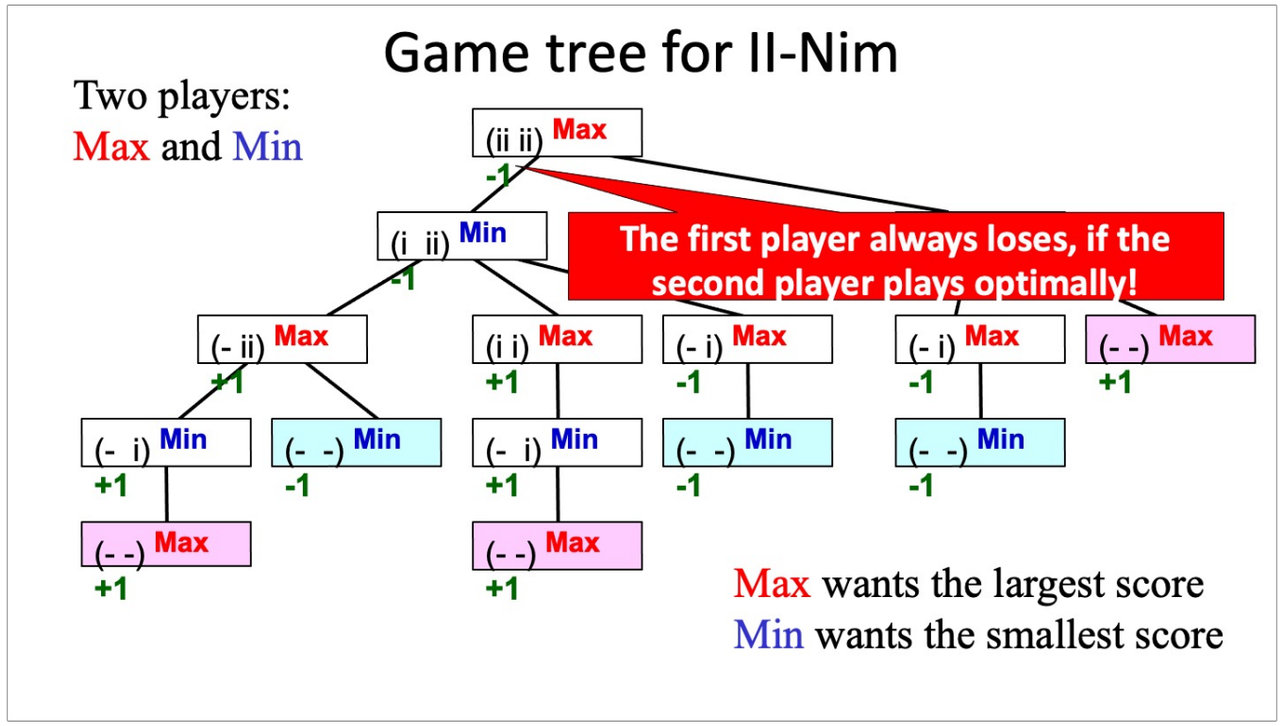
Max wants the largest score
Min wants the smallest score

Game tree for II-Nim

Two players:
Max and Min



Max wants the largest score
Min wants the smallest score



Interesting conclusion: the first play will always lose if the second player plays optimally. From the rule of the game we don't immediately see who will win. However, with the game tree and the game-theoretical value, we can obtain the highly non-trivial conclusion that the first play will always lose if the second player plays optimally.

Two other important implications:
1. We can compute the game-theoretical values easily from bottom up.
2. Once we have the values of the children of a current state, then we know which is the best action. That is, to play the game, all we need is to compute the values of the current state.

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**

The bottom-up approach needs to have the entire tree. This requires too large space complexity. We can then use the DFS idea to address this, which leads to the minimax algo.

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)

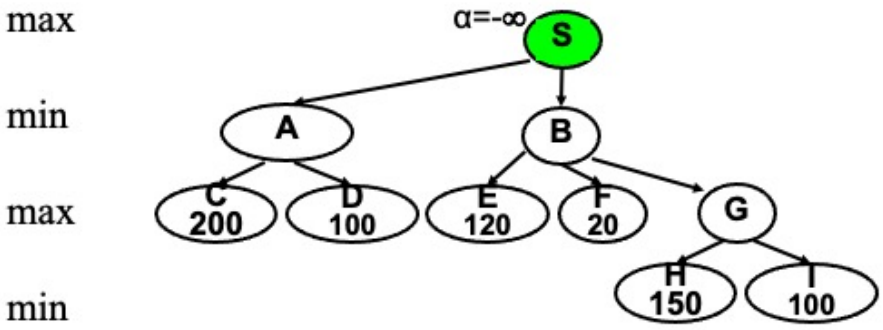
The minimax algorithm replaces the bottom-up computation with recursion.

The key idea of recursion: we assume that smaller problems are already solved, and we want to use the solutions for the smaller problems to solve the current problem. Here, the smaller problems are the values of the children, and the current problem is the value of the current state.

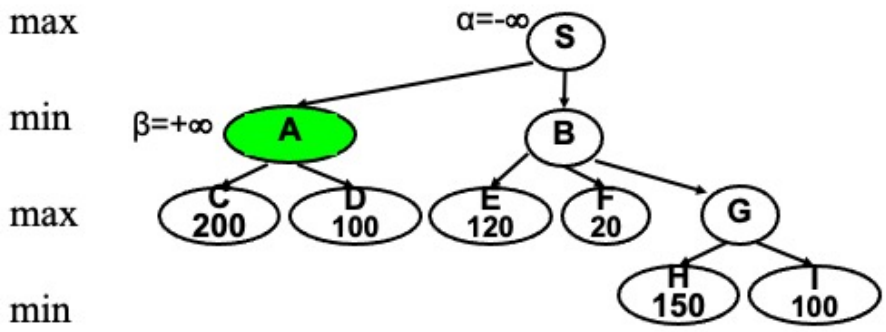
On a state where Max is going to play:
1. If it's terminal then we can return the terminal score which is the value by definition
2. If not terminal, just take the maximum of the values of the children (here we pretend that we have already solve the smaller problems of computing the values of the children)

This is the Max-Value function. Similar for the Min-Value function that computes the value of a state where Min is going to play.

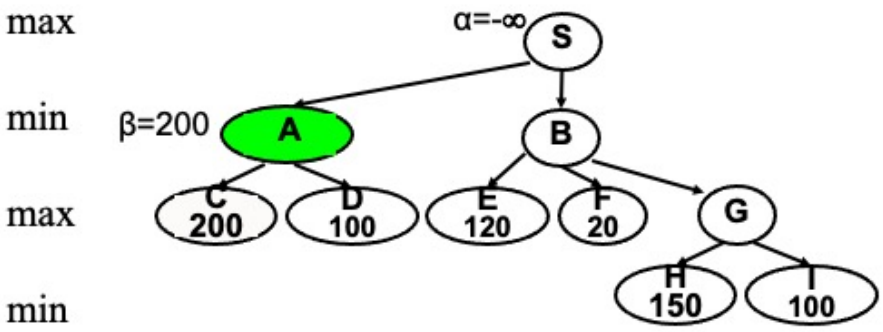
Minimax algorithm in execution



Minimax algorithm in execution

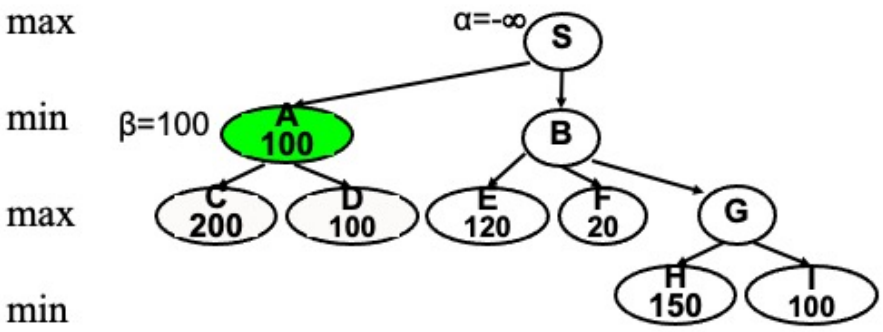


Minimax algorithm in execution

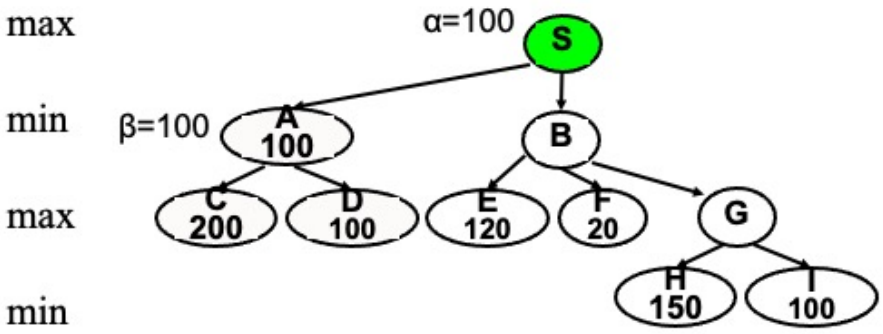


The execution on the terminal nodes is omitted.

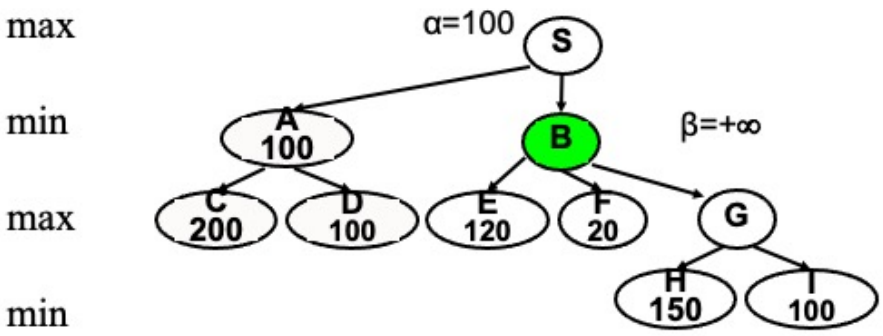
Minimax algorithm in execution



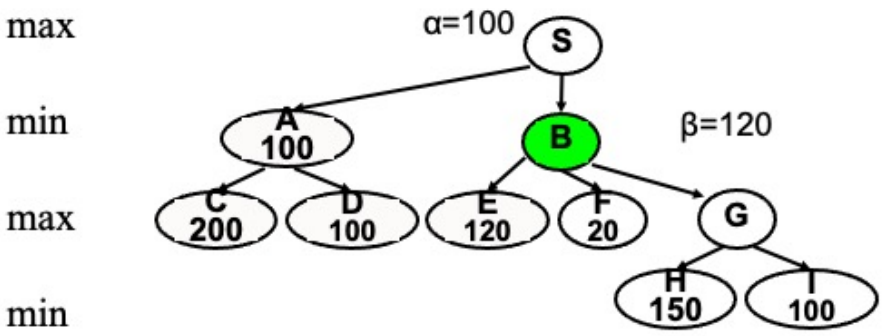
Minimax algorithm in execution



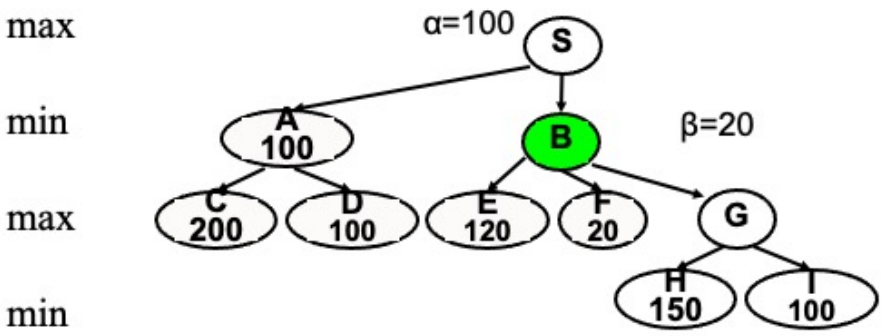
Minimax algorithm in execution



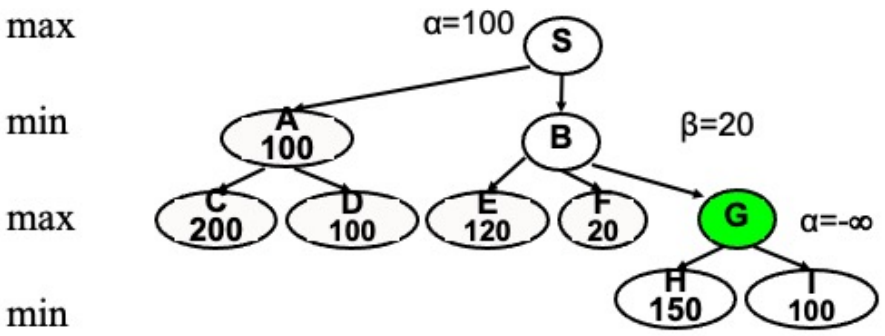
Minimax algorithm in execution



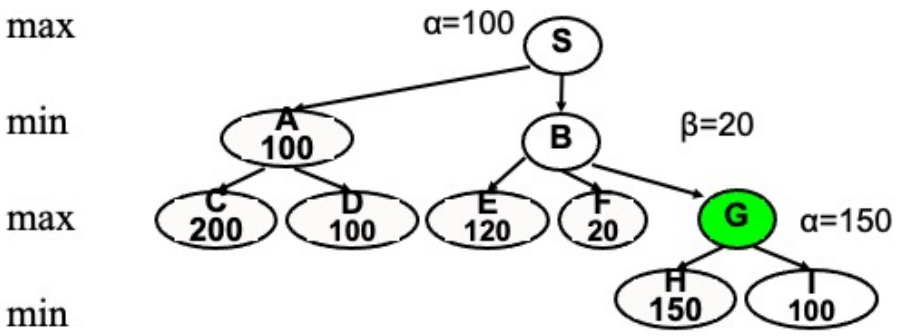
Minimax algorithm in execution



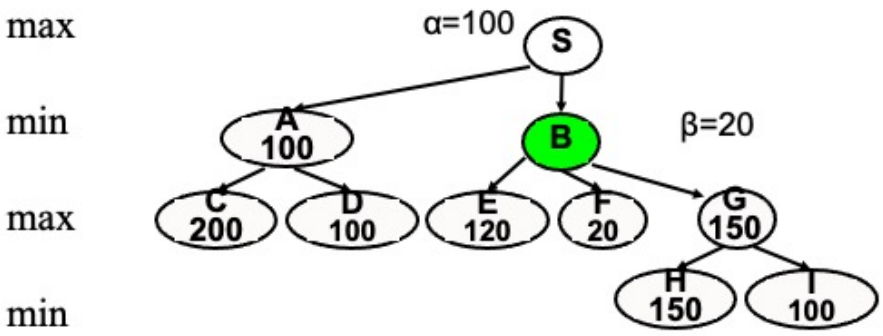
Minimax algorithm in execution



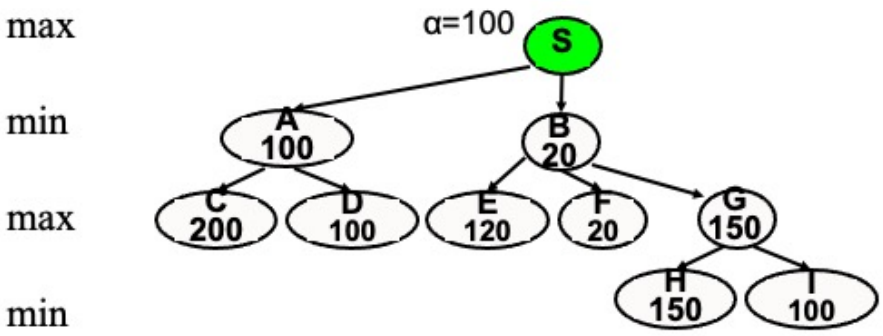
Minimax algorithm in execution



Minimax algorithm in execution



Minimax algorithm in execution



Minimax 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)*

The time complexity of the minimax algo is not good: exponential in m, the number of steps.

We can address this by limiting the search depth, similar to what we have done in iterative deepening. That is, if we want to compute the value of a current state, we only go down the current state for depth d.

The question is: what if we get to a node at depth d but it's not a terminal state? What value should we return? We can just use some estimation.

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
```

The modified minimax:

1. We keep a depth budget count. Each time we go down one step in the recursion, we discount the depth budget by 1
2. We stop at terminal state or when we exhausted the depth budget along the search path.

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

We can combine the Max-Value and Min-Value functions into one function, since they are very similar.

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)

A common way to design the heuristic function: linear model, which is a weighted sum of some designed features.

The features are typically some intuitive important information about the state, like the difference of white and black pieces in Chess.
The weights are set according to the importance of the features.

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).