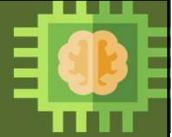


Elective Course

Course Code: CS4103

Autumn 2025-26



Lecture #12

Artificial Intelligence for Data Science

Week-3: PROBLEM SOLVING BY SEARCH

Introduction to Informed Search [Part-IV] &
Adversarial Search--- Games [Part-I]

Course Instructor:

Dr. Monidipa Das

Assistant Professor

Department of Computational and Data Sciences

Indian Institute of Science Education and Research Kolkata, India 741246

Simulated Annealing Algorithm



function SIMULATED-ANNEALING(*problem*, *schedule*) **returns** a solution state

inputs: *problem*, a problem

schedule, a mapping from time to “temperature”

current \leftarrow MAKE-NODE(*problem*.INITIAL-STATE)

for $t = 1$ **to** ∞ **do**

$T \leftarrow$ *schedule*(t)

if $T = 0$ **then return** *current*

next \leftarrow a randomly selected successor of *current*

$\Delta E \leftarrow$ *next*.VALUE – *current*.VALUE

if $\Delta E > 0$ **then** *current* \leftarrow *next*

else *current* \leftarrow *next* only with probability $e^{\Delta E/T}$

Simulated Annealing Search



- Probability of a move decreases with the amount ΔE by which the evaluation is worsened
- A second parameter T is also used to determine the probability: high T allows more worse moves, T close to zero results in few or no bad moves
- **Schedule** input determines the value of T as a function of the completed cycles

Dr. Monidipa Das, Department of CDS, IISER Kolkata

Simulated Annealing Search



- **Idea:** improve hill-climbing by allowing occasional down-hill steps, to minimize the probability of getting stuck in a local maximum
- Down-hill steps taken randomly but with probability that decreases with time
- Probability controlled by a given annealing schedule for a temperature parameter T
- If schedule lowers T slowly enough, search is guaranteed to end in a global maximum
- **Catch:** it may take several tries with test problems to devise a good annealing schedule
- **Simulated Annealing vs. Hill Climbing**

Dr. Monidipa Das, Department of CDS, IISER Kolkata



Adversarial Search--- Games

Dr. Monidipa Das, Department of CDS, IISER Kolkata



Search vs. Games

- **Search- No adversary**
 - Solution is a path from start to goal, or a series of actions from start to goal
 - Heuristics and search techniques can find optimal solution
 - Evaluation function: estimate of cost from start to goal through given node
 - Actions have cost
 - Example: Path planning, Scheduling activities
- **Games- Adversary**
 - Solution is strategy
 - Strategy specifies move for every possible opponent reply
 - Time limits force an approximate solution
 - Evaluation function: evaluate “goodness” of game position
 - Board configurations have utility
 - Examples: Chess, Checkers, Othello, Backgammon

Dr. Monidipa Das, Department of CDS, IISER Kolkata

Why Study Games in AI?



- Problems are formalized
- Real world knowledge (common sense knowledge) is not too important
- Rules are fixed
- Adversary modeling is of general importance
 - opponent introduces uncertainty
 - programs must deal with the contingency problem
- Complexity of games?
 - number of nodes in a search tree (e.g., 10^{40} legal positions in chess)
 - specification is simple (no missing information, well-defined problem)

Dr. Monidipa Das, Department of CDS, IISER Kolkata

Types of games



	Deterministic	Chance
Perfect Information	Chess, Checkers, Othello, Go	Backgammon, Monopoly
Imperfect Information		Bridge, Poker

Not considered: Physical games like tennis, croquet, ice hockey, etc.

We restrict our analysis to a very specific set of games:

2-player zero-sum discrete finite deterministic games of perfect information

Dr. Monidipa Das, Department of CDS, IISER Kolkata

2-player zero-sum discrete finite deterministic games of perfect information



- **What does it mean?**
 - **2-player:** Two players involved (multi-agent environment)
 - **Zero-sum:** In any outcome of any game, Player A's gains equal player B's losses.
 - **Discrete:** All game states and decisions are discrete values.
 - **Finite:** Only a finite number of states and decisions.
 - **Deterministic:** No chance (no die rolls).
 - **Perfect information:** Both players can see the state, and each decision is made sequentially (no simultaneous moves).

Dr. Monidipa Das, Department of CDS, IISER Kolkata

Games as Adversarial Search



- **States:**
 - board configurations
- **Initial state:**
 - the board position and which player will move
- **Successor function:**
 - returns list of (action, state) pairs, each indicating a legal move and the resulting state
- **Terminal test:**
 - determines when the game is over
- **Utility function:**
 - gives a numeric value in terminal states (e.g., -1, 0, +1 for loss, tie, win)

Dr. Monidipa Das, Department of CDS, IISER Kolkata

Evaluation function



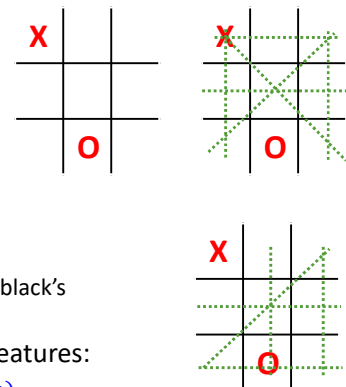
- **Evaluation function** or **static evaluator** is used to evaluate the “goodness” of a game position.
 - Contrast with heuristic search where the evaluation function was a non-negative estimate of the cost from the start node to a goal and passing through the given node
- The **zero-sum assumption** allows us to use a single evaluation function to describe the goodness of a board with respect to both players.
 - $f(n) \gg 0$: position n good for player A and bad for player B
 - $f(n) \ll 0$: position n bad for player A and good for player B
 - $f(n)$ near 0: position n is a neutral position
 - $f(n) = +1$: win for player A
 - $f(n) = -1$: win for player B

Dr. Monidipa Das, Department of CDS, IISER Kolkata

Evaluation function examples



- Example of an evaluation function for **Tic-Tac-Toe**:
 - $f(n) = [\text{\#of win path for player A}] - [\text{\#of win path for player B}]$
 where a win path is a complete row, column, or diagonal
 [E.g. $f(n) = 6 - 5 = 1$ for the state given in this slide.
 Here, player A gives cross and player B naught]
- Alan Turing's function for **chess**:
 - $f(n) = w(n)/b(n)$
 where $w(n)$ = sum of the point value of white's pieces and $b(n)$ = sum of black's
- Most evaluation functions are specified as a weighted sum of position features:
 - $f(n) = w_1 * \text{feature1}(n) + w_2 * \text{feature2}(n) + \dots + w_k * \text{featurek}(n)$
 Example features for chess are piece count, piece placement, squares controlled, etc.
- Deep Blue had over 8000 features in its evaluation function



Dr. Monidipa Das, Department of CDS, IISER Kolkata

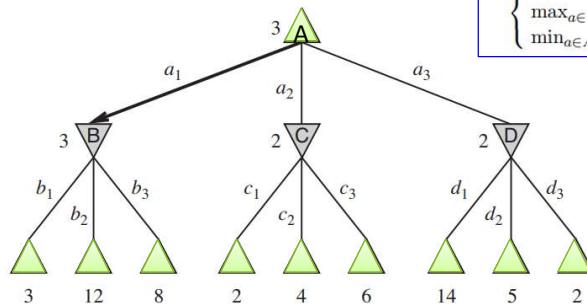
Minimax



- Perfect play for deterministic games
- **Idea:** choose move to position with highest minimax value
= best achievable payoff against best play
- E.g., 2-ply game:

MAX

MIN



$$\text{MINIMAX}(s) = \begin{cases} \text{UTILITY}(s) & \text{if } \text{TERMINAL-TEST}(s) \\ \max_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if } \text{PLAYER}(s) = \text{MAX} \\ \min_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if } \text{PLAYER}(s) = \text{MIN} \end{cases}$$

Dr. Monidipa Das, Department of CDS, IISER Kolkata

Minimax algorithm Adversarial analogue of DFS



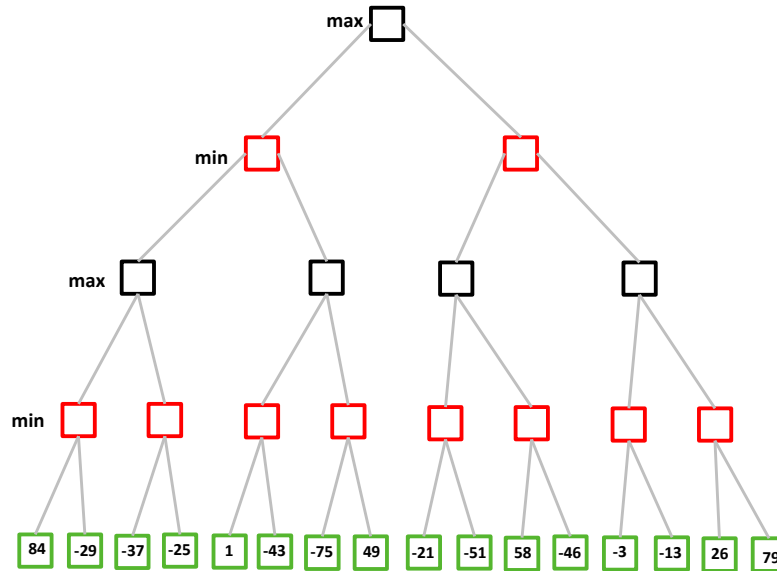
```
function MINIMAX-DECISION(state) returns an action
  return arg maxa ∈ ACTIONS(s) MIN-VALUE(RESULT(state, a))
```

```
function MAX-VALUE(state) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← -∞
  for each a in ACTIONS(state) do
    v ← MAX(v, MIN-VALUE(RESULT(s, a)))
  return v
```

```
function MIN-VALUE(state) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← ∞
  for each a in ACTIONS(state) do
    v ← MIN(v, MAX-VALUE(RESULT(s, a)))
  return v
```

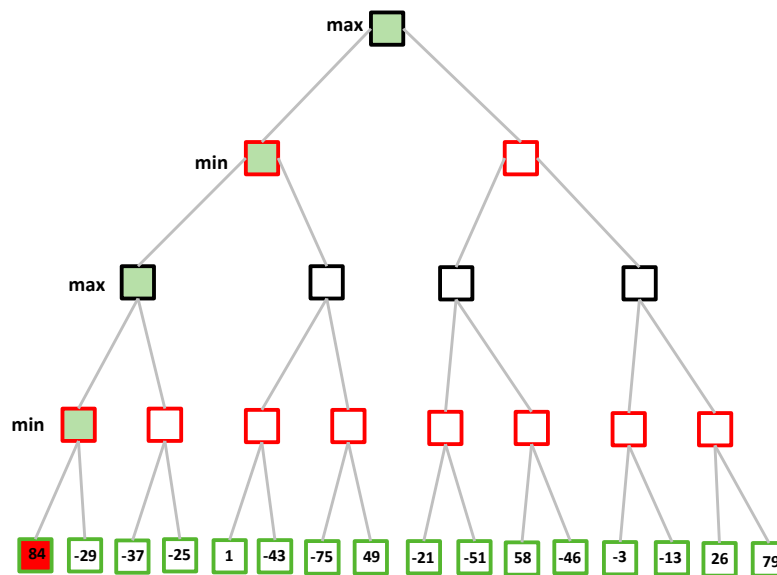
Dr. Monidipa Das, Department of CDS, IISER Kolkata

Example



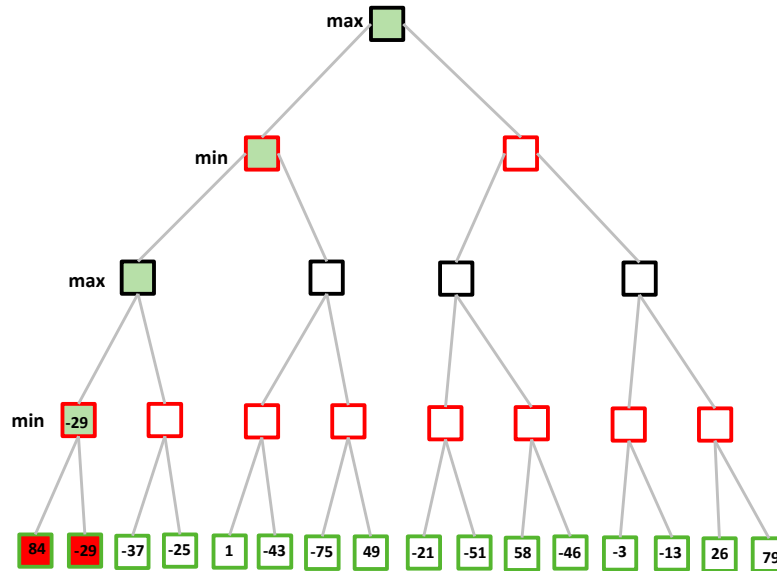
Dr. Monidipa Das, Department of CDS, IISER Kolkata

Example



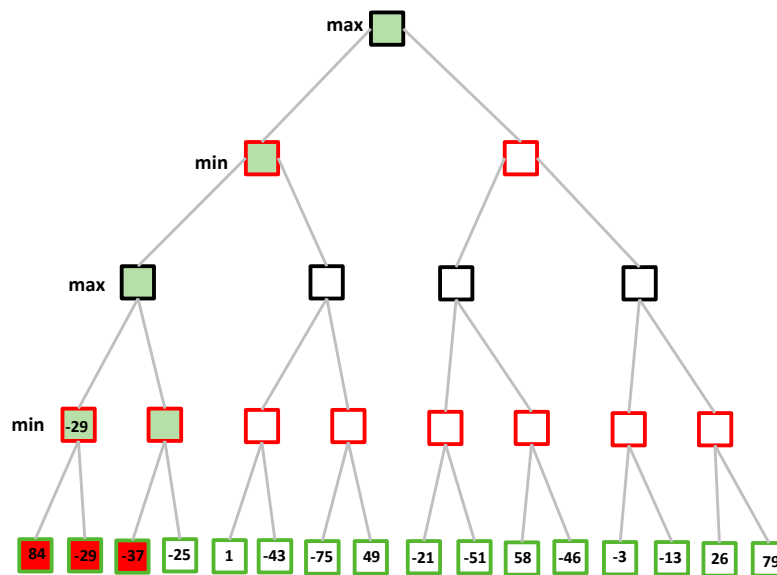
Dr. Monidipa Das, Department of CDS, IISER Kolkata

Example



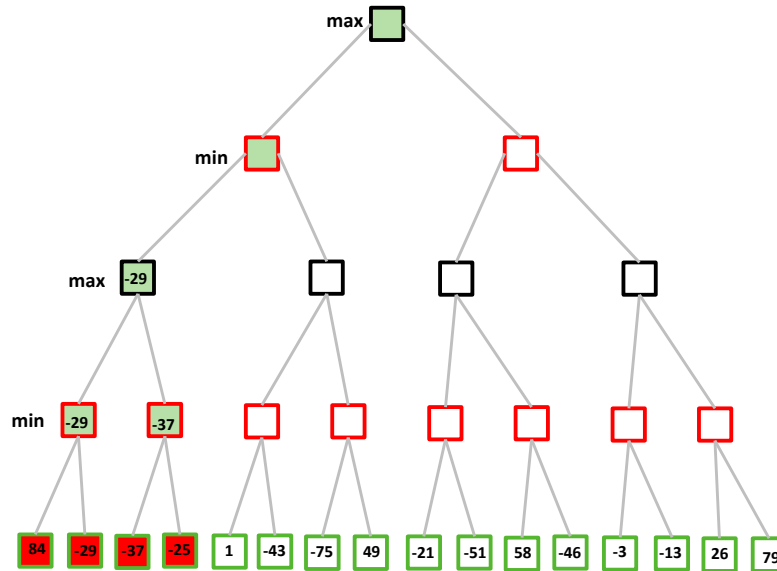
Dr. Monidipa Das, Department of CDS, IISER Kolkata

Example



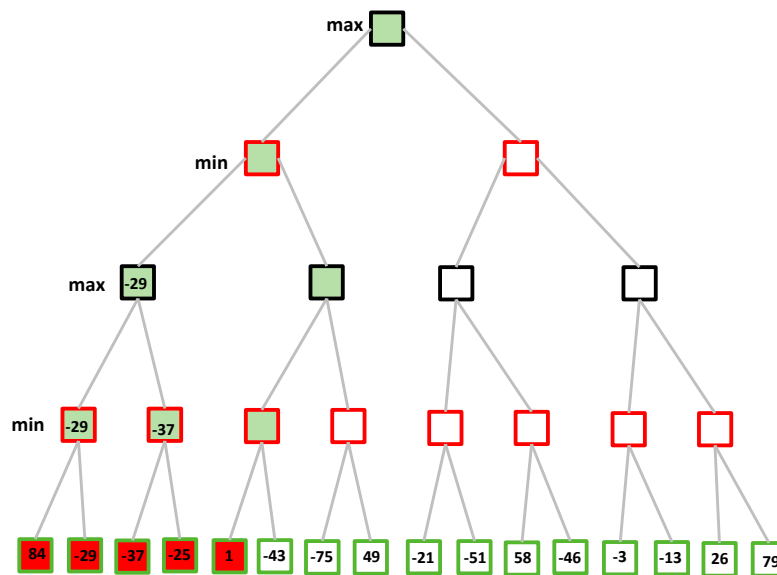
Dr. Monidipa Das, Department of CDS, IISER Kolkata

Example



Dr. Monidipa Das, Department of CDS, IISER Kolkata

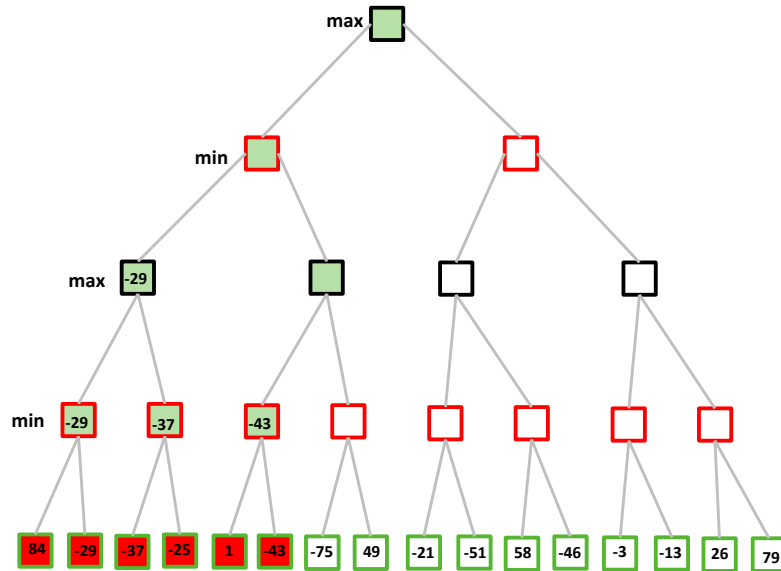
Example



Dr. Monidipa Das, Department of CDS, IISER Kolkata

Example

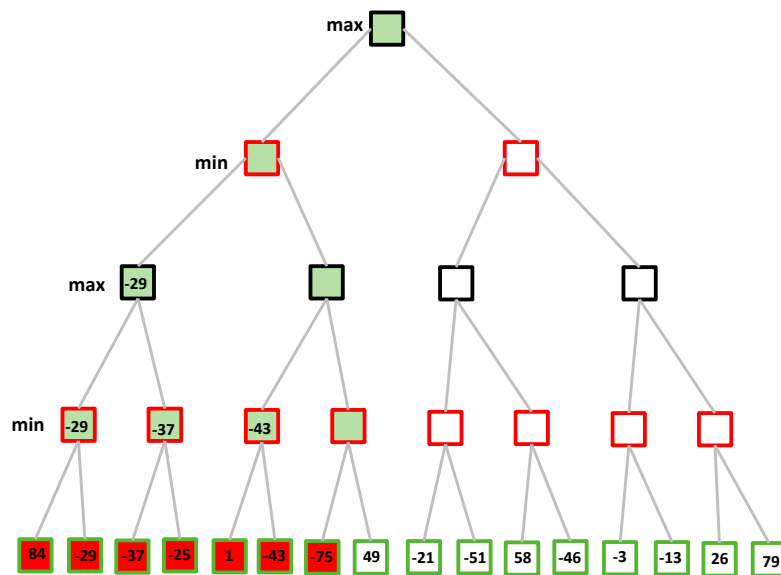
23



Dr. Monidipa Das, Department of CDS, IISER Kolkata

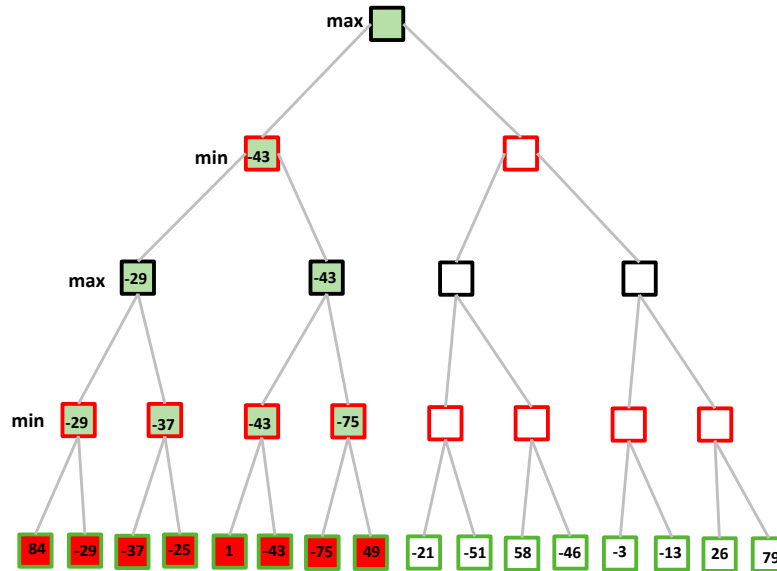
Example

24



Dr. Monidipa Das, Department of CDS, IISER Kolkata

Example

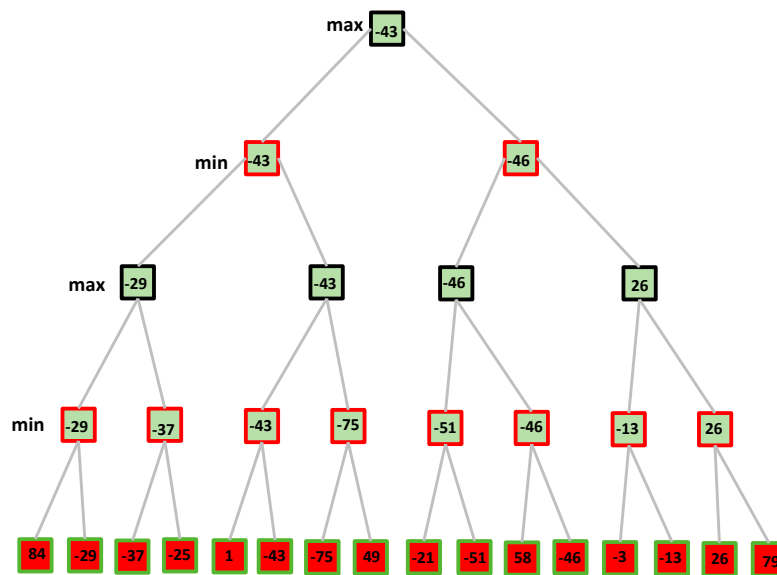


Dr. Monidipa Das, Department of CDS, IISER Kolkata

Example

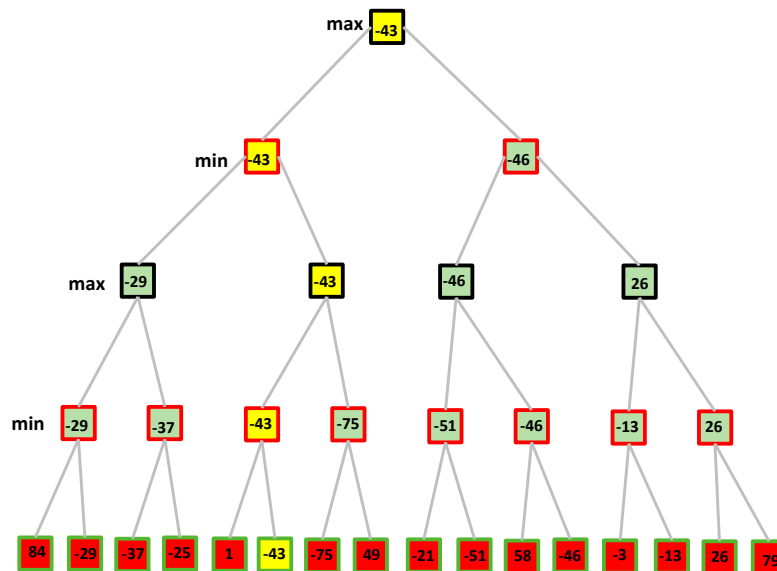


...similarly



Dr. Monidipa Das, Department of CDS, IISER Kolkata

Example



Dr. Monidipa Das, Department of CDS, IISER Kolkata

Properties of Minimax



- **Complete?**
 - Yes (if tree is finite)
- **Optimal?**
 - Yes (against an optimal opponent)
 - Can it be beaten by an opponent playing sub-optimally?
 - No
- **Time Complexity?**
 - $O(b^m)$
- **Space Complexity?**
 - $O(bm)$ [depth-first search, generate all actions at once]
 - $O(m)$ [backtracking search, generate actions one at a time]

Dr. Monidipa Das, Department of CDS, IISER Kolkata

Good Enough?



- **Tic-Tac-Toe:**
 - branching factor $b \approx 5$ (on average)
 - game length $m \approx 9$
 - search space $\approx 5^9 \approx 1,953,125$
 - Exact solution quite reasonable
- **Chess:**
 - branching factor $b \approx 35$
 - game length $m \approx 100$
 - search space $\approx 35^{100} \approx 10^{154}$
 - **Exact solution completely infeasible**

Dr. Monidipa Das, Department of CDS, IISER Kolkata



Questions?

Dr. Monidipa Das, Department of CDS, IISER Kolkata