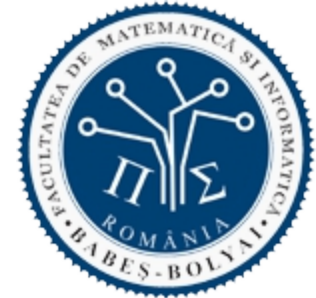




BABEȘ-BOLYAI UNIVERSITY  
Faculty of Computer Science and Mathematics



# ARTIFICIAL INTELLIGENCE

**Solving search problems**

Adversarial search

# Topics

---

A. Short introduction in Artificial Intelligence (AI)

**B. Solving search problems**

**A. Definition of search problems**

**B. Search strategies**

A. Uninformed search strategies

B. Informed search strategies

C. Local search strategies (Hill Climbing, Simulated Annealing, Tabu Search, Evolutionary algorithms, PSO, ACO)

**D. Adversarial search strategies**

**C. Intelligent systems**

A. Rule-based systems in certain environments

B. Rule-based systems in uncertain environments (Bayes, Fuzzy)

C. Learning systems

A. Decision Trees

B. Artificial Neural Networks

C. Support Vector Machines

D. Evolutionary algorithms

D. Hybrid systems

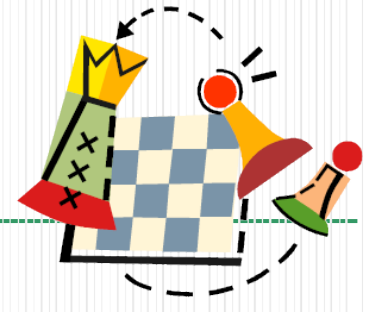
# Useful information

---

- Chapter II.5 of *S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall, 1995*
- Chapter 6 of *H.F. Pop, G. Șerban, Inteligență artificială, Cluj Napoca, 2004*
- Documents from folder *05\_adversial\_minimax*

# Content

## Games



- A bit of history
- Some theoretical aspects
- Games and search
  - Definitions, components, topology
  - Strategies and search algorithms

# Games – a bit of history

---

- ❑ Challenges of intelligence
  - Checkers (Mesopotamia, cc. 3000 î.Hr.)
  - Go (China, sec. VI î.Hr.)
  - Chess (India, sec. VI d.Hr.)
  
- ❑ Games as search algorithms
  - Classical search – a single agent that tries, without obstacles, to meet its target
  - Games – search in the presence of an adversary (opponent agent) that comes with some uncertainty
  
- ❑ Games and AI
  - Algorithm design and testing
  - One the oldest domain of AI

# Games – a bit of history

---

## □ Game playing

### ■ By humans

- Intelligent activity
- Comparing the abilities

### ■ By computers

- Favorable environment for developing AI techniques
  - Game = a problem that is
    - Structured (success or fail)
    - Solvable by search (simple search or heuristic search)
  - Limits

# Games – some theoretical aspects

---

- ❑ Difficult problems with minimal initial knowledge
- ❑ States and actions easier to be represented
- ❑ Few knowledge about environment is required
- ❑ Games are a particular case of search problems
- ❑ Frequently, they have huge search spaces
  - Game's complexity → uncertainty
    - ❑ Due to insufficient time for computing the consequences of all possible moves
    - ❑ Not due to lack of information
- ❑ Game are interesting 😊

# Games and search

---

- Formalism
- Game's representation
- Typology
- Search space
  - Representation
  - Exploration methods



# Games and search – formalism

---

- Solving a search problem
  - Stage  $x$ : definition of the search space
    - Linear spaces
    - Tree-based spaces
      - Trees
      - Graphs
  - Stage  $x + 1$ : selection of a search strategy
    - Which move/strategy is the best?
      - That wins the game
      - That can be quickly identified (time complexity is reduced)
      - That can be identified by minimal resources (space complexity is reduced)
- Problem:
  - How the best next move can be identified as soon as possible?
- One solution:
  - Search among possible moves and their consequences

# Games and search – formalism

---

- Adversarial search is used in games with players that try to maximise their score, but they have one or more opponents
- Game's representation as search problems
  - **States**
    - Board configurations + the player that has to move
    - All the possible states → search tree
  - **Operators (actions, successor functions)**
    - Allowed moves
  - **Initial state**
    - Initial game configuration
  - **Final state**
    - (winning) configuration that ends the game
  - **Utility function (score function)**
    - Associates a numerical value to each state
- Difficulties
  - Games can be difficult problems
    - Simple to be formalised
  - Optimal solution identification can be infeasible
    - A winning solution is acceptable
    - An unacceptable solution has large costs and could cause the defeat

# Games and search – formalism

---

## □ Definitions

### ■ Conflict

- Situation when more parts act and have contrary purposes
- The consequence of a part' action depends on the actions of other parts

### ■ Game

- Simplified modelling of a conflict
- Set of decisions (actions, moves) of the parts

### ■ Move

- A function  $H : \{\text{game' positions}\} \rightarrow \{\text{game' positions}\}$
- If  $p$  – a game position,  $H(p)$  – a new position

### ■ Rules

- System of conditions that regard the possible moves

### ■ Strategy

- Rules that define the free moves of the game for a given game' state

# Games and search – representation

---

## □ Components

- Players
- Moves (strategies)
- Winning criteria

## □ Game representations

- Strategic form → matrix
  - Players
  - Strategies
  - Profits for each player and each strategy
- Expanded form → tree
  - Level → player
  - Node → move

# Games and search – representation



## □ Strategic form → matrix

- Eg. Prisoner's Dilemma: two prisoners are questioned by the police. Police knows something about what they did, but it has not all the information. To find out, the two prisoners are detained in two cells and are questioned. Prisoners have two options:
  - can tell the whole story (can betray one another)
  - can not say anything (can cooperate)
- No prisoner knows what the other will respond. Depending on their answers, the penalties are:
  - If both silent (cooperates), the sentence is easy (one year each)
  - If one talks (betray) and one silent (cooperates), the betrayer is released, and the defendant gets 10 years
  - If both talk (betray each other), the sentence is 5 years each
- What will the two prisoners do?

P1 \ P2	Cooperation	Betrayal
Cooperation	(1,1)	(10,0)
Betrayal	(0,10)	(5,5)

# Games and search – representation



## □ Strategic form $\rightarrow$ matrix

- Eg. Hunting deer: two guys go hunting. Each can choose to hunt:
  - a deer or
  - a rabbit
- without knowing what the other person chose.
  - Hunting deer is more profitable (gain = 4) and can only be done in two.
  - Hunting rabbits is less valuable (gain = 1) can be done individually.
- What they choose to hunt?

P1 \ P2	Deer	Rabbit
Deer	(4,4)	(1,3)
Rabbit	(3,1)	(3,3)

# Games and search – representation

## □ Strategic form → matrix

### ■ Eg. economic

- Publicity of two companies in the same market
- Agreements level cartel pricing

### ■ Eg. Sport



- Two cyclists are in the front row wearing platoon train (cooperate) to not be overtaken. Often, only one train goes (co) and the finish line is betrayed by his opponent.

### ■ Ex. sociological



When you meet a new person, we tend to be very careful to have a safe position (competition). Both may indicate a desire to move from defensive positions to

- interaction and
- recognition of a common intention

# Games and search – typology

---

- Number of players
  - 1
  - 2
  - More
  
- Communication between players
  - Cooperative
  - Non-cooperative
  
- Game's strategies
  - Symmetric
  - Asymmetric



# Games and search – typology

---

## □ Player's profit

### ■ Zero sum

- Profit of a player = loss of other players → a player must win or the game ends in a draw

### ■ Non-zero sum

- Profit of a player  $\neq$  loss of other players

## □ Nature of performed moves

### ■ Free moves

- A move is selected from the set of possible moves → deterministic games

### ■ Random moves

- Random factor (dice, playing cards, coins) → non-deterministic games

# Games and search – typology

---

## □ Game's information

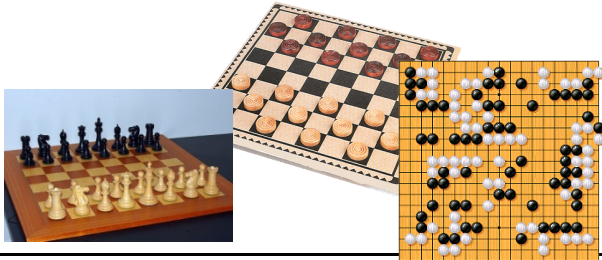
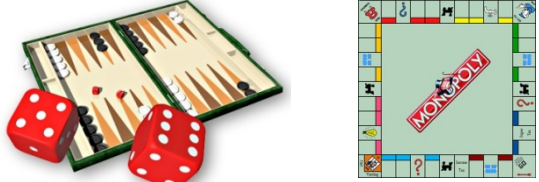

### ■ Perfect information

- A player, before to move, knows the consequences of all the other previous moves (its moves and the moves of the other players)
- Usually, the game involves sequential moves

### ■ Imperfect information

- A player does not know all the previous moves

# Games and search – typology

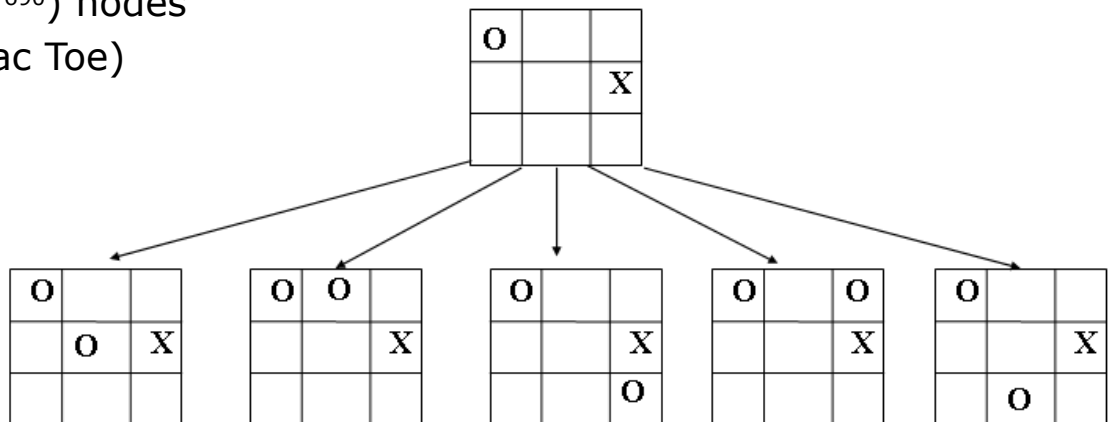
	Deterministic	Non-deterministic (random)
Perfect information		
Imperfect information	Vaporașe/Avioane/ Submarine	

# Games and search – search space



## □ Representation

- Linear space
- Tree-based space → game's tree
  - Identify the winning strategy → explore the entire tree
  - Very large for some games
    - Chess (AI drosophila)
      - Ramification factor  $\approx 35$
      - $\approx 50$  moves/player
      - $\approx 35^{100}$  ( $10^{154}$ ) nodes
      - $10^{40}$  different nodes
    - Go
      - $\approx 200$  moves/state, 300 levels
      - $200^{300}$  ( $10^{690}$ ) nodes
  - Example – XO (Tic Tac Toe)



# Games and search – search space

---



## □ Strategy (for a player)

### ■ Definition

- All the moves performed by a player
- Based on
  - Game's rules
  - Current state of game
- $\neq$  move

### ■ Typology

- Aim
  - Step-by-step strategies
    - At each step, the optimal move is identified
  - Complete strategies
    - A sequence of moves is identified

# Games and search – search space

---



## □ Game's strategy

### ■ Step by step

- Ex.: XO, Checkers, Chess
- Algorithms – can work by:
  - Linear structures
    - Strategy of symmetry
    - Strategy of pairs
    - Parity strategy
    - Dynamic programming
    - Other strategies
  - Tree-based structures
    - AndOr trees
    - MiniMax (with Alpha-Beta cuttings)

### ■ Complete

- Ex.: labyrinth, map navigation
- Algorithm can identify
  - An optimal path between 2 locations
  - A sequence of actions for moving the player from a location into another location

# Games and search – search space



## □ Game's strategy

### ■ Step by step

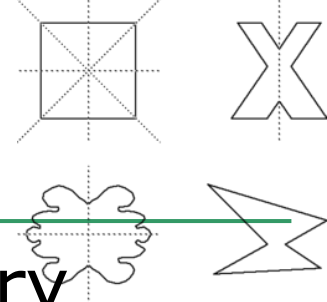
- Ex.: XO, Checkers, Chess
- Algorithms – can work by:
  - **Linear structures**
    - Strategy of symmetry
    - Strategy of pairs
    - Parity strategy
    - Dynamic programming
    - Other strategies
  - Tree-based structures
    - AndOr trees
    - MiniMax (with Alpha-Beta cuttings)

### ■ Complete

- Ex.: labyrinth, map navigation
- Algorithm can identify
  - An optimal path between 2 locations
  - A sequence of actions for moving the player from a location into another location

# Games and search – search space

Game's strategy → strategy of symmetry

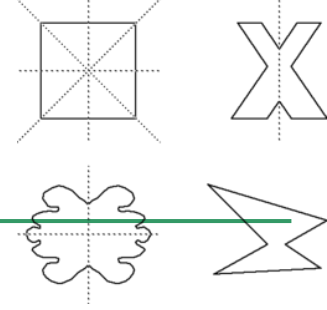


## □ Main idea

- Player B imitates player A based on a symmetry axis (center)
- If A can move, then B can move also
  - Game ends when A can not move
- It is possible that the first move to not have symmetric

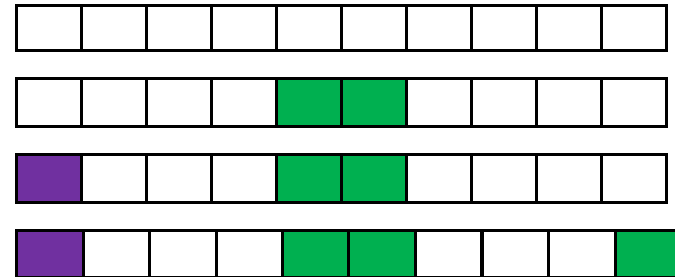


# Games and search – search space

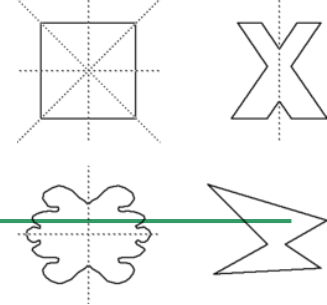


## Game's strategy → strategy of symmetry

- Eg. Game fill the cells
  - We have:
    - A strip of paper is divided into  $N$  cells. Alternatively two players A and B each fill maximum  $k$  adjacent empty cells ( $n$  and  $k$  have the same parity). One who can not move loses. Originally moves the player A.
  - It requires:
    - To determine if the player A has a winning strategy. If so, to program moves of A, the B's being read from the keyboard.
  - Study case:
    - $n = 10, k = 4$

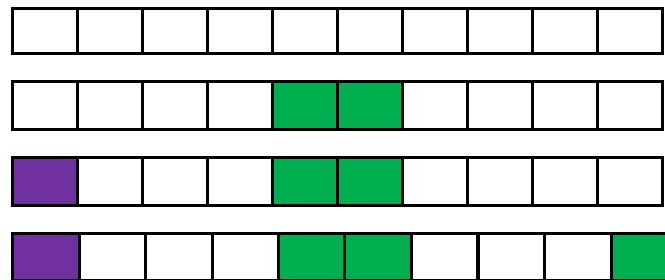


# Games and search – search space



## Game's strategy → strategy of symmetry

- Eg. Game fill the cells
  - We have:
    - A strip of paper is divided into  $N$  cells. Alternatively two players A and B each fill maximum  $k$  adjacent empty cells ( $n$  and  $k$  have the same parity). One who can not move loses. Originally moves the player A.
  - It requires:
    - To determine if the player A has a winning strategy. If so, to program moves of A, the B's being read from the keyboard.
  - Study case:
    - $n = 10, k = 4$



- Solution
  - if  $n$  – even
    - First move → player A fill an even # of cells in the middle of the strip
    - For next moves, player A imitates the moves of player B (taking into account the middle of the strip)
  - If  $n$  - odd
    - First move → player A fill an odd # of cells in the middle of the strip
    - For next moves, player A imitates the moves of player B (taking into account the middle of the strip)

# Games and search – search space



## Game's strategy → strategy of pairs

### □ Main idea

- Generalisation for strategy of symmetry
- Groups the moves in pairs
  - (move of player A, move of player B)
- If A can move, then B can move also
  - Game ends when A can not move
- It is possible that the first move does not have a pair

# Games and search – search space



## Game's strategy → strategy of pairs

### □ Eg. *Sliding squares*

#### ■ It gives:

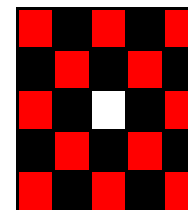
- Each square of a table  $n * n$  ( $n$  - odd) is red or black so the board resembles a checker board. Box in the middle of the sheet is empty (does not contain a square). Alternatively, two players A (red squares) and B (black squares) move one of their squares on the board free square. Square must be initially moved into a free neighbour square (horizontal or vertical). The player that can not move any of his squares loses the game. Player B moves first.

#### ■ It requires:

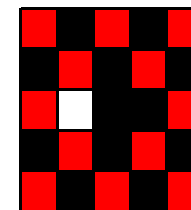
- Determine if the player A has a winning strategy. If so, program moves to A, the B's being read from the keyboard.

#### ■ Study case:

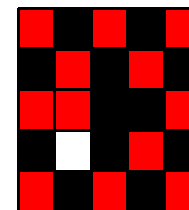
- $n = 5$  (initial game proposed by G.W.Lewthwaite)



a



b



c

# Games and search – search space



## Game's strategy → strategy of pairs

### □ Eg. *Sliding squares*

#### ■ It gives:

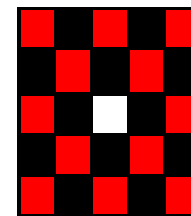
- Each square of a table  $n * n$  ( $n$  - odd) is red or black so the board resembles a checker board. Box in the middle of the sheet is empty (does not contain a square). Alternatively, two players A (red squares) and B (black squares) move one of their squares on the board free square. Square must be initially moved into a free neighbour square (horizontal or vertical). The player that can not move any of his squares loses the game. Player B moves first.

#### ■ It requires:

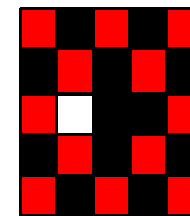
- Determine if the player A has a winning strategy. If so, program moves to A, the B's being read from the keyboard.

#### ■ Study case:

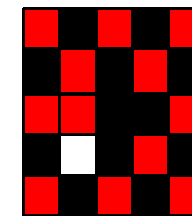
- $n = 5$  (initial game proposed by G.W.Lewthwaite)



a



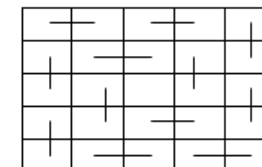
b



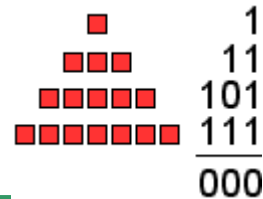
c

#### ■ Solution

- Split the table in Dominoes, the square of the middle (initially free) does not belong to a domino
- After each move of player B, the free square belongs to a domino that covers a red square also. This red square will be moved by player A for its next move (player A has always a square to move)
- Covering the board by dominoes starts from upper-left corner and follows a spiral way



# Games and search – search space



## Game's strategy – parity strategy

### □ Main idea

#### ■ 2 positions for a game

- Singular position (even position)
- Non-singular position (odd position)

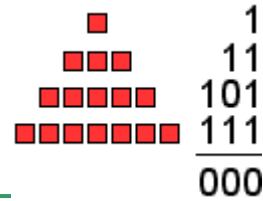
#### ■ Theorems

- T1: an odd position is transformed, by a suitable move, into an even position (player A)
- T2: an even position is transformed, by any move, into an odd position (player B)

#### ■ Winner

- → end position = even position

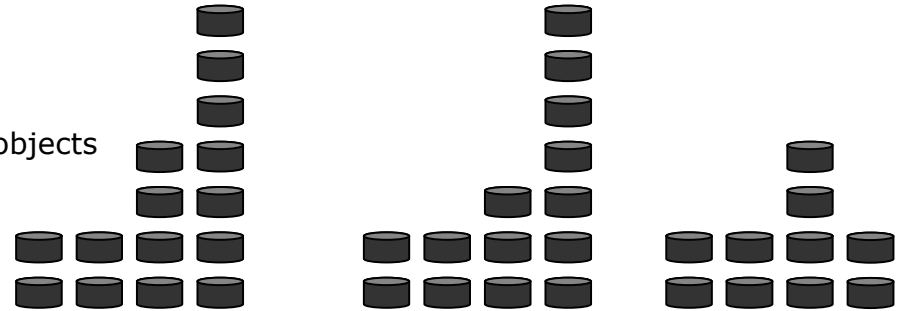
# Games and search – search space



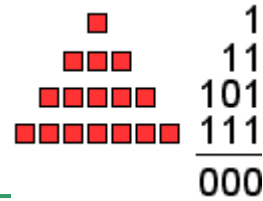
## Game's strategy – parity strategy

### □ Eg. – NIM game

- It gives:
  - N stacks, each with  $p_i$  objects and 2 players A and B that, alternatively, extracts from a single stack a number of objects. The player that performed the last move wins the game. Player A starts the game.
- It requires:
  - Determine if player A has a winning strategy. If it has, program its moves, while B's are read from the keyboard.
- study case:
  - 4 stacks with 2, 2, 4 and 7, respectively, objects



# Games and search – search space



## Game's strategy – parity strategy

### □ Eg. – NIM game

#### ■ It gives:

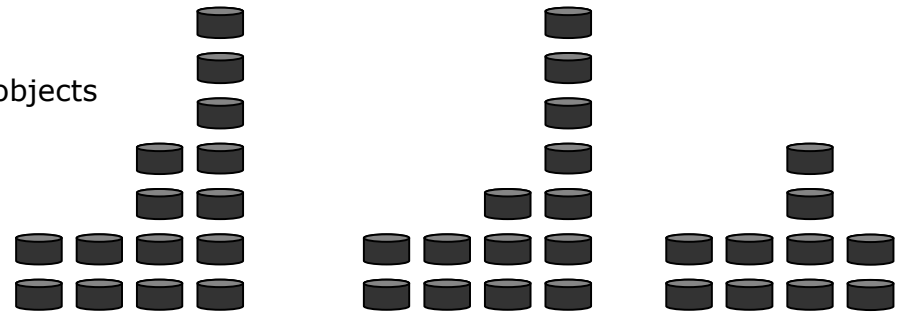
- N stacks, each with  $p_i$  objects and 2 players A and B that, alternatively, extracts from a single stack a number of objects. The player that performed the last move wins the game. Player A starts the game.

#### ■ It requires:

- Determine if player A has a winning strategy. If it has, program its moves, while B's are read from the keyboard.

#### ■ study case:

- 4 stacks with 2, 2, 4 and 7, respectively, objects

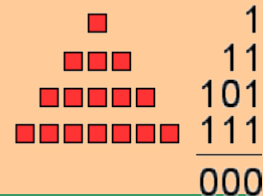


#### ■ Solution

- # of objects of each stack can be binary represented (as sum of 2's powers)
  - Each natural number admits a unique decomposition as a sum of 2's powers
- Even state – all 2's powers from game representation appear for an even number of times (on all stacks)
- Off state – any state that is not even
  - T1 → an odd position is transformed, by a suitable move, into an even position (player A)
  - T2 → an even position is transformed, by any move, into an odd position (player B)
- Select the stack that contains the most significant bit on position k
  - k – position of the most representative bit of the NIM sum (for all the stacks)



# Games and search – search space



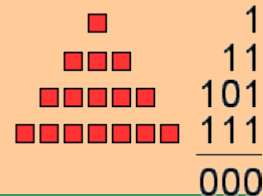
## Game's strategy – parity strategy

### □ Eg. – NIM game

#### ■ Theorem proving

- $S_1: 2 = 2^1, S_2: 3 = 2^1 + 2^0, S_3: 4 = 2^2, S_4: 5 = 2^2 + 2^0$
- $2^0 - 2, 2^1 - 2, 2^2 - 2 \rightarrow$  even state
- Extract from a stack with an even number of objects
  - An even # of objects (eg. 2 objects from  $S_3$ ) = >
    - $S_1: 2 = 2^1, S_2: 3 = 2^1 + 2^0, S_3: 4-2 = 2^1, S_4: 5 = 2^2 + 2^0$
    - $2^0 - 2, 2^1 - 3, 2^2 - 1 \rightarrow$  odd state
  - An odd # of objects (eg. 1 object from  $S_1$ )
    - $S_1: 2-1 = 2^0, S_2: 3 = 2^1 + 2^0, S_3: 4 = 2^2, S_4: 5 = 2^2 + 2^0$
    - $2^0 - 3, 2^1 - 1, 2^2 - 2 \rightarrow$  odd state
- Extract from a stack with an odd number of objects
  - An even # of objects (eg. 2 objects from  $S_4$ ) = >
    - $S_1: 2 = 2^1, S_2: 3 = 2^1 + 2^0, S_3: 4 = 2^2, S_4: 5-2 = 2^1 + 2^0$
    - $2^0 - 2, 2^1 - 3, 2^2 - 1 \rightarrow$  odd state
  - An odd # of objects (eg. 1 object from  $S_2$ )
    - $S_1: 2 = 2^1, S_2: 3-1 = 2^1, S_3: 4 = 2^2, S_4: 5 = 2^2 + 2^0$
    - $2^0 - 1, 2^1 - 2, 2^2 - 2 \rightarrow$  odd state

# Games and search – search space



## Game's strategy – parity strategy

### □ Eg. – NIM game

#### ■ Algorithm

##### □ Suppose the game:

- $S_1: 3 = 2^1 + 2^0 \rightarrow 011$ ,  $S_2: 4 = 2^2 \rightarrow 100$ ,  $S_3: 5 = 2^2 + 2^0 \rightarrow 101$

##### □ Steps:

##### 1. Compute the NIM sum ( $S_{\text{Nim}}$ ) for all the stacks

- $S_{\text{Nim}} = S_1 \text{ XOR } S_2 \text{ XOR } S_3 = 3 \text{ XOR } 4 \text{ XOR } 5 = 011 \text{ XOR } 100 \text{ XOR } 101 = 010 = 2$

##### 2. Identify the position $k$ of the most significant 1 in the NIM sum

- $S_{\text{Nim}} = 2 = 010_{(2)} \rightarrow$  second position ( $k = 2$ )

##### 3. Search the stack that contains the most significant bit on position $k$ (the stack that decreases if XOR is applied between it and the NIM sum)

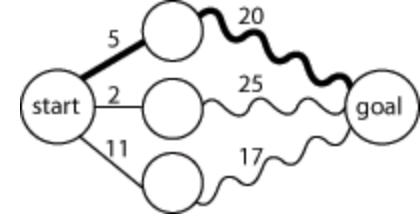
- $S_1: 3 = 010$ ,  $S_2: 4 = 100$ ,  $S_3: 5 = 101 \Rightarrow S_1$  or
- $S_1 \text{ XOR } S_{\text{Nim}} = 3 \text{ XOR } 2 = 011 \text{ XOR } 010 = 001 = 1$
- $S_2 \text{ XOR } S_{\text{Nim}} = 4 \text{ XOR } 2 = 100 \text{ XOR } 010 = 110 = 6$
- $S_3 \text{ XOR } S_{\text{Nim}} = 5 \text{ XOR } 2 = 101 \text{ XOR } 010 = 111 = 7 \rightarrow S_1$

##### 4. Extract from this stack some objects such as the other stacks form an even state; # of object that remains on that stack = that stack XOR NIM sum

- $S_1 \text{ XOR } S_{\text{Nim}} = 3 \text{ XOR } 2 = 011 \text{ XOR } 010 = 001 = 1 \rightarrow 2 (=3 - 1)$  objects are taken from stack 1

##### 5. Goto step 1

# Games and search – search space



## Game's strategy – dynamic programming (DP)

### □ Main idea

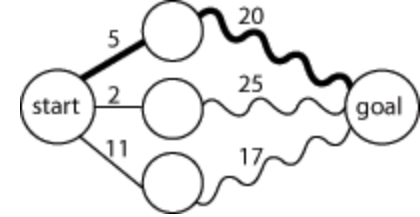
#### ■ Steps for problem solving by DP:

- Decompose the problem in more sub-problems
- Solve the sub-problems
- Combine the sub-solutions in order to obtain the final solution

#### ■ Steps for game solving by DP:

- Decomposed the game in more sub-games
- Identify a perfect winning strategy for each sub-game
- Combine the sub-strategies in order to obtain the final strategy

# Games and search – search space



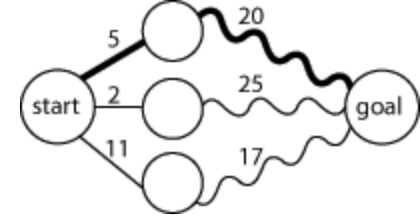
## Game's strategy – dynamic programming (DP)

### □ Main idea

#### ■ How a game is solved by DP?

- Decompose the game in sub-games  $G_h$  (lowest level sub-games) and
  - Label each game  $G_h$  by T (true) or F (false) based on the probability of the player (that has to move) to have a winning strategy
- A sub-game  $G_k$  from an interior level ( $k < h$ ) will be labelled by:
  - T, if there is at least a sub-game  $G_i$ ,  $k < i$ , labelled by F, that can be reached by a move from game  $G_k$
  - F, if all sub-games  $G_i$ ,  $k < i$ , that can be reached by a move from the game  $G_k$  are tagged with T

# Games and search – search space



## Game's strategy – dynamic programming (DP)

### □ Eg. – array of cells

#### ■ It gives:

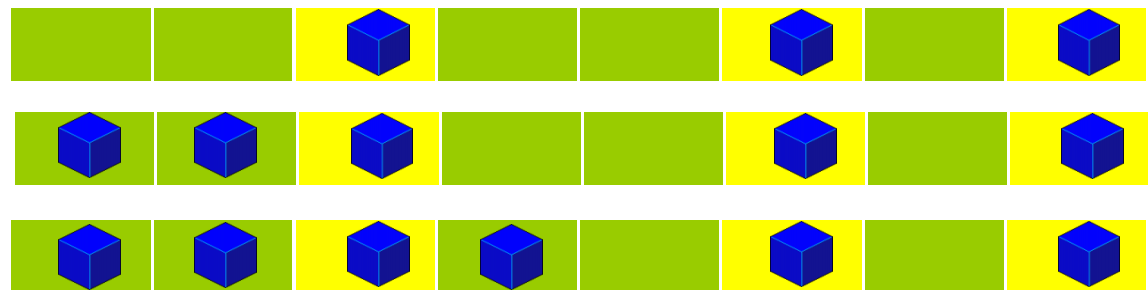
- An array of  $n$  cells that can contain cubes (maximum one cube in a cell). The goal of the game is to fill all the cells by cubes. 2 players, alternatively, fill by cubes (complete or partially) the most left sub-array of empty cells. The player that can not move loses the game. First player is A.

#### ■ It requires:

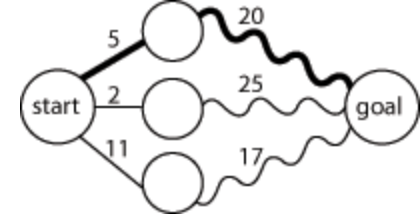
- Determine if the player A has a winning strategy, and if it has, that program the A's moves, that of B's being read from keyboard.

#### ■ Example

- $n = 8$



# Games and search – search space



Game's strategy – dynamic programming (DP)

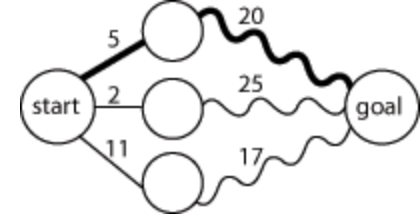
□ Eg. – array of cells

■ Solution:



T

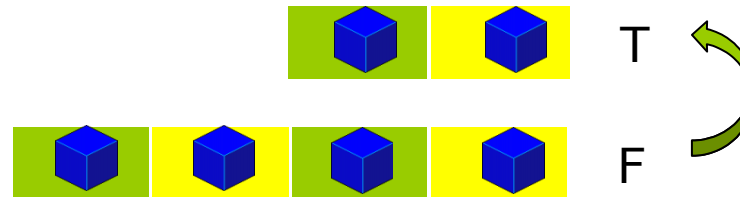
# Games and search – search space



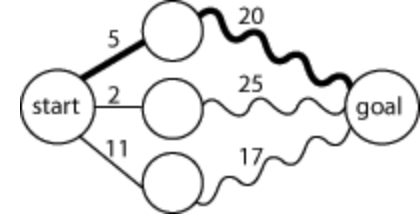
Game's strategy – dynamic programming (DP)

□ Eg. – array of cells

■ Solution:



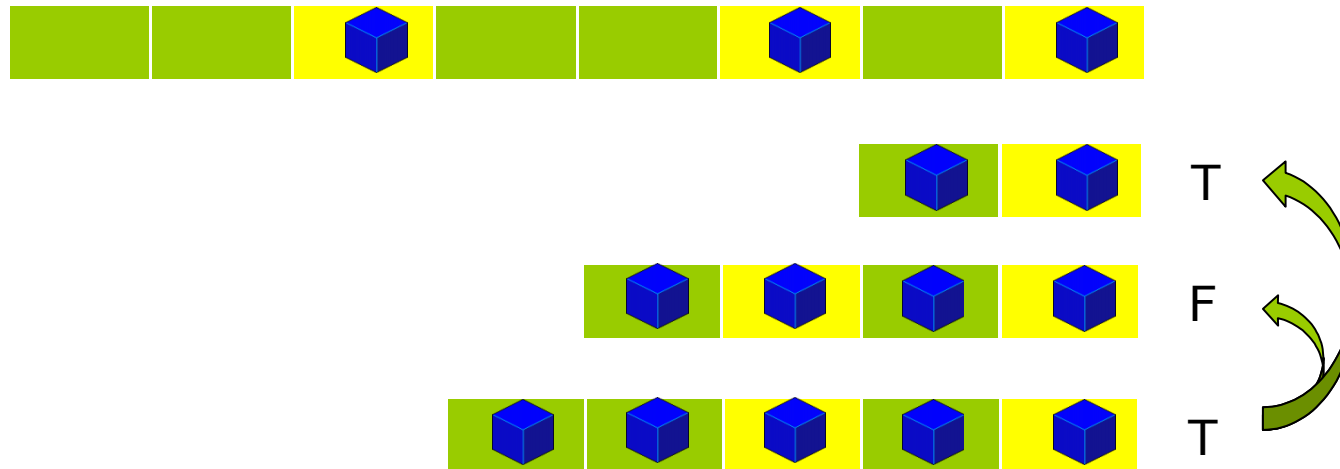
# Games and search – search space



Game's strategy – dynamic programming (DP)

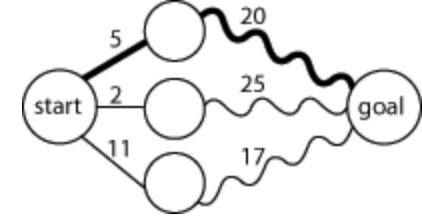
□ Eg. – array of cells

■ Solution:





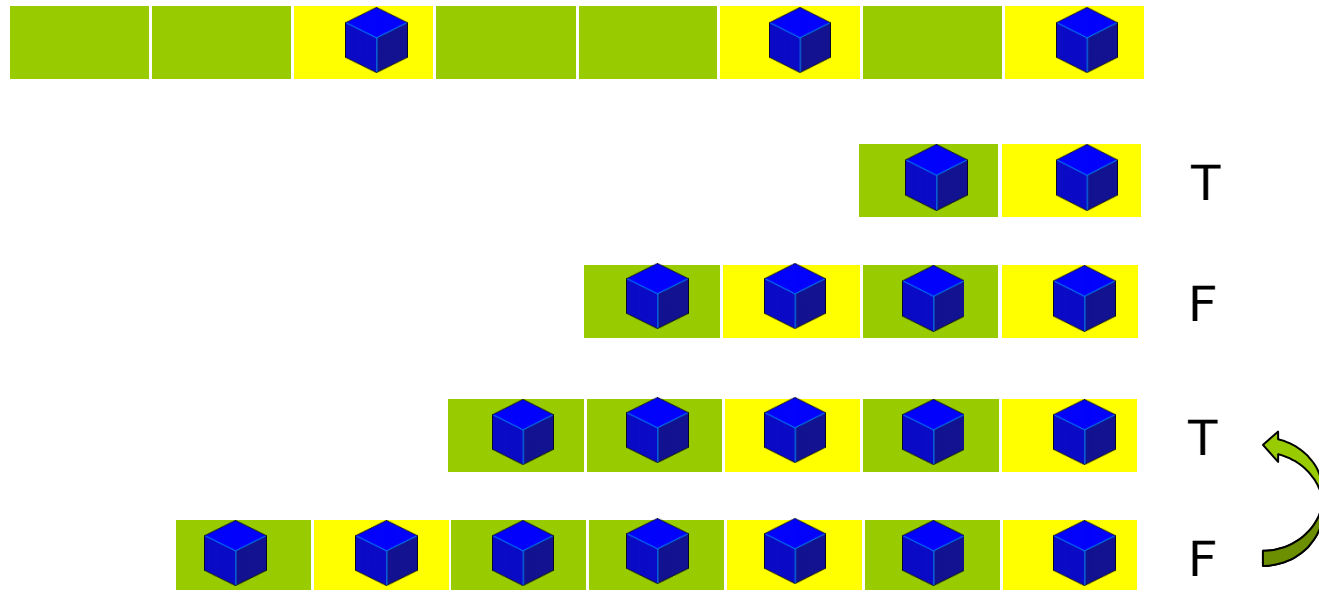
# Games and search – search space



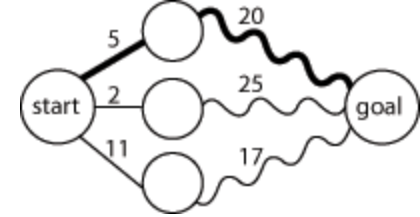
Game's strategy – dynamic programming (DP)

□ Eg. – array of cells

■ Solution:



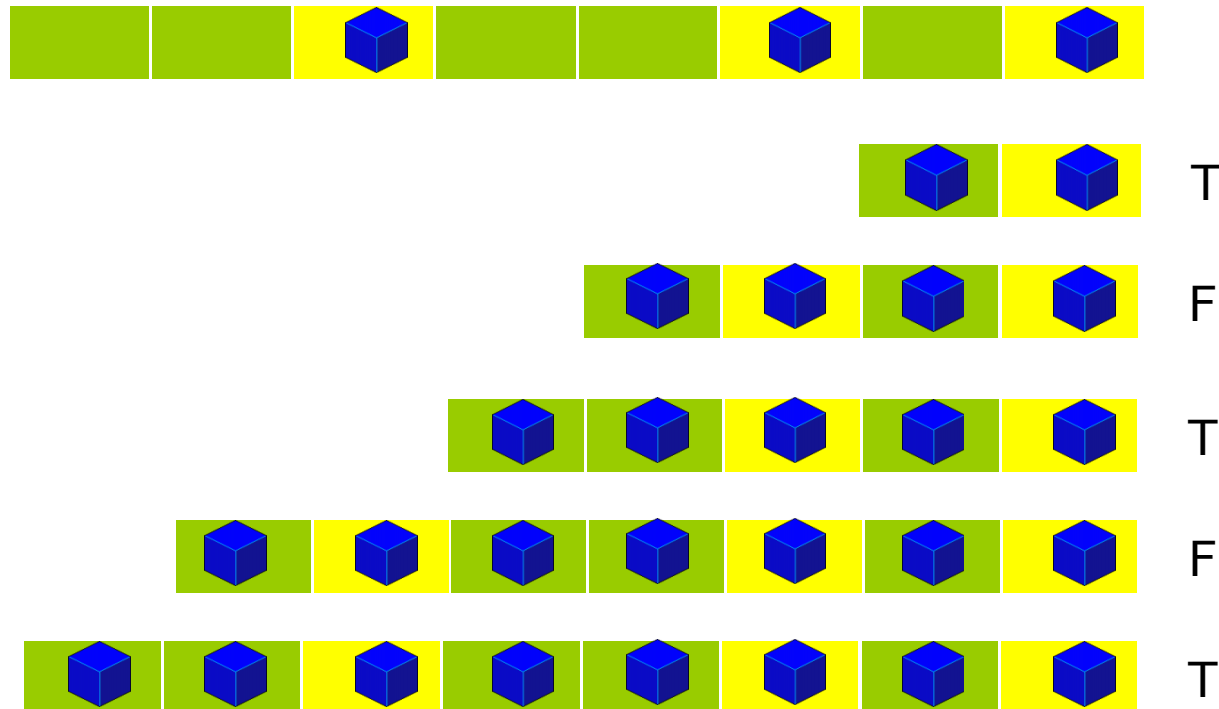
# Games and search – search space



Game's strategy – dynamic programming (DP)

□ Eg. – array of cells

■ Solution:



# Games and search – search space



## □ Game's strategy

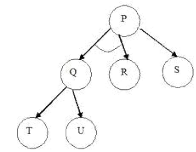
### ■ Step by step

- Ex.: XO, Checkers, Chess
- Algorithms – can work by:
  - Linear structures
    - Strategy of symmetry
    - Strategy of pairs
    - Parity strategy
    - Dynamic programming
    - Other strategies
  - **Tree-based structures**
    - **AndOr trees**
    - MiniMax (with Alpha-Beta cuttings)

### ■ Complete

- Ex.: labyrinth, map navigation
- Algorithm can identify
  - An optimal path between 2 locations
  - A sequence of actions for moving the player from a location into another location

# Games and search – search space



## Game's strategies – AndOr trees (AOT)

### □ Main idea

#### ■ **Search space representation** by help of AOTs for a 2 players game

##### □ OR nodes

- → select a move for the current player A
- → there is a possibility (a move) for player A to move in an AND node

##### □ AND nodes

- → consider all the possible moves of the opponent
- → by any move, player B moves in an OR node

##### □ Other meaning:

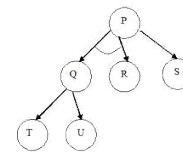
- Root → problem of the winning player (that starts the game)
- Possible moves for player A are combined by dis-junction (OR)
- Possible moves of player B are combine by conjunction (AND)

##### □ The game can be won if it starts by an OR node

#### ■ **Difficulties**

##### □ To large trees

# Games and search – search space



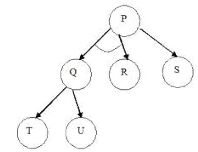
## Game's strategies – AndOr trees (AOT)

### □ Main idea

#### ■ Steps for solving a game by AOTs

- Decompose the game in more sub-games
  - Nodes on more levels
  - Nodes of a level correspond to the possible moves of a player
- Identify a perfect winning strategy for each sub-game (node)
  - Label the nodes by T or F, depending on the possibility of player A to have a winning strategy for that sub-game
  - Labeling rules
    - Leaves are labeled by T or F depending on game configuration
    - Internal nodes are labeled by:
      - T (for player A) if there is at least one child node labeled by T – OR rule
      - T (for player B) if all the children are labeled by T – AND rule
- Combine the sub-strategies in order to obtain the final strategy

# Games and search – search space



## Game's strategies – AndOr trees (AOT)

### □ Example

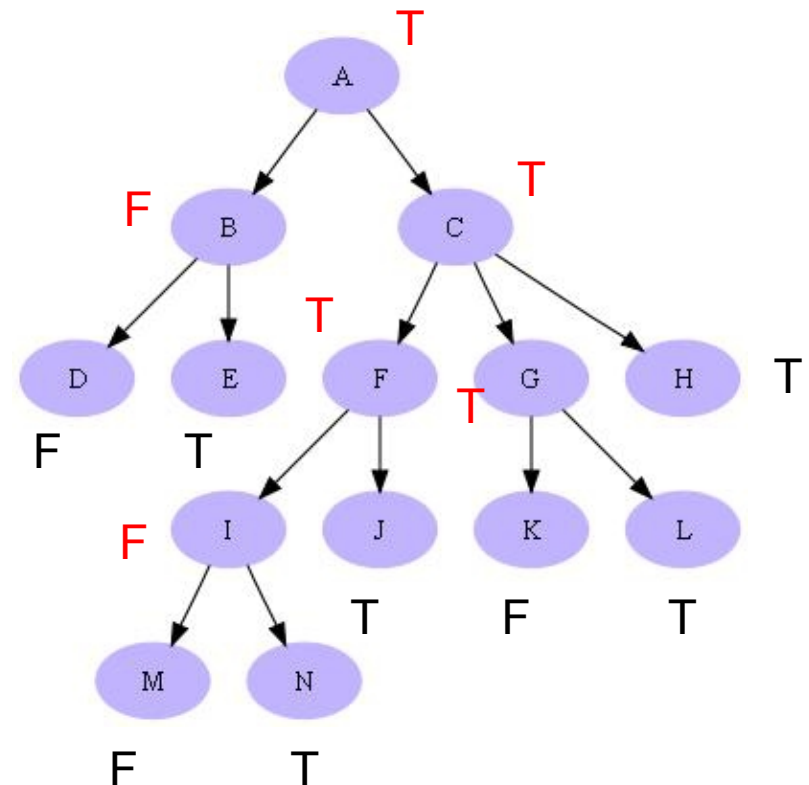
A (OR)

B (AND)

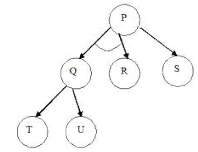
A (OR)

B (AND)

A (OR)



# Games and search – search space

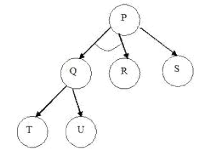


## Game's strategies – AndOr trees (AOT)

### □ Algorithm

```
bool backAndOr(Node N, int level){  
    //level = 0 for the node in the top of the tree.  
    if (N is a terminal) {  
        if (the first player won)  
            return true;  
        else  
            return false;  
    }  
    else{  
        if (level % 2){           //B is about to move; AND  
            result = true;  
            for each child  $N_i$  of N  
                result = result && backAndOr( $N_i$ , level+1);  
        }  
        else{                     // A is about to move; OR  
            result = false;  
            for each child  $N_i$  of N  
                result = result || backAndOr( $N_i$ , level+1);  
        }  
        return result;  
    }  
}
```

# Games and search – search space



## Game's strategies – AndOr trees (AOT)

- Advantages
  - Can be applied for solving any game
- Disadvantages
  - Require a large memory
  - Require a large computing time
- ➔ mini-max algorithm



# Games and search – search space



## □ Game's strategy

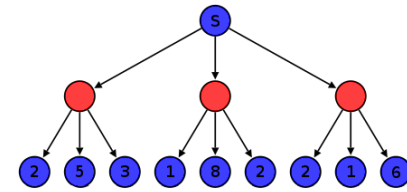
### ■ Step by step

- Ex.: XO, Checkers, Chess
- Algorithms – can work by:
  - Linear structures
    - Strategy of symmetry
    - Strategy of pairs
    - Parity strategy
    - Dynamic programming
    - Other strategies
  - **Tree-based structures**
    - AndOr trees
    - **MiniMax** (with Alpha-Beta cuttings)

### ■ Complete

- Ex.: labyrinth, map navigation
- Algorithm can identify
  - An optimal path between 2 locations
  - A sequence of actions for moving the player from a location into another location

# Games and search – search space

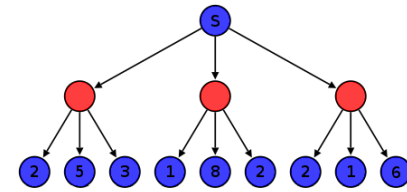


## Game's strategies → MiniMax

### □ Theoretical aspects

- Proposed by John von Neuman in 1944
- Main idea: maximize the position of a player, while the position of the opponent is minimizing
- The search tree = levels for profit maximization of one player's profit alternate with levels for profit minimization of the opponent
  - First player tries to maximize its profit (MAX player -> first move)
  - Second player tries to minimize the gain of the first player (MIN player → the opponent)
- Identify the best moves
  - Construct the tree for all the possible moves (a move for each state of the game)
  - Evaluate the leaf (where a player wins)
  - Propagate down-to-top the profits
  - Tree exploration respects DFS method
- Generate all the moves
  - Possible only for simple games (ex. Tic-Tac-Toe)
  - Impossible for complex games

# Games and search – search space



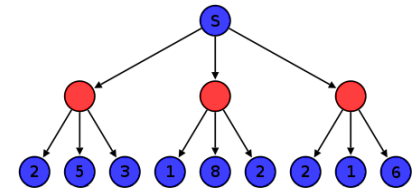
## Game's strategies → MiniMax

### □ Algorithm

- Construct the tree of all possible moves
- Evaluate the leaf
- While it is possible to select a node for whose children were evaluated
  - Choose a node
  - If the node is on a Min level, then it will be evaluated to the value of the minimum children
  - If the node is on a Max level, then it will be evaluated to the value of the maximum children
- Return the root's value (root is on a Max level)

```
int backMiniMax(Node N, int level){
    //level = 0 for the node in the top of the tree.
    if (level == MaxLevel)
        return the quality of N computed with a heuristic;
    else if (level < MaxLevel){
        if (level % 2){
            //B is about to move; minimize
            result = MaxInt;
            for each child Ni of N
                result = minim(result, backMiniMax(Ni, level+1));
        }
        else{
            // A is about to move; Maximize
            result = -MaxInt;
            for each child Ni of N
                result = maxim(result, backMiniMax(Ni, level+1));
        }
        return result;
    }
}
```

# Games and search – search space



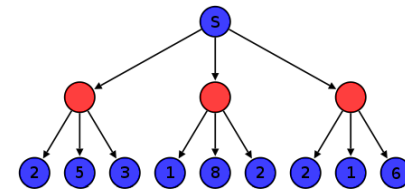
Game's strategies → MiniMax

□ Example - *Tic-Tac-Toe* game

■ Evaluation of a node:

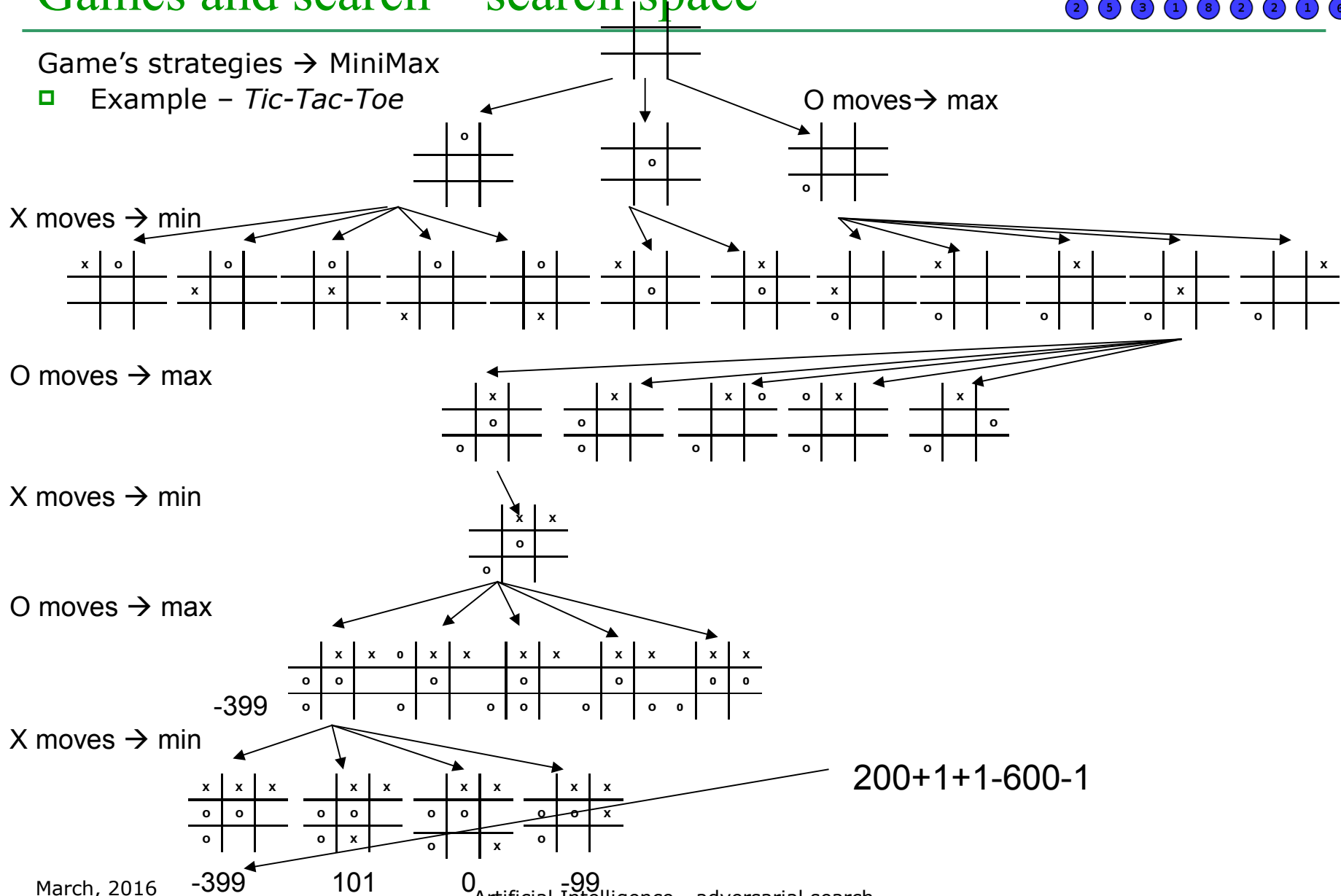
- If a line is almost complete (2 equal signs) → 200 points
- If there are 2 almost complete lines (2 equal signs) → 300 points
- A complete line → 600 points
- A potential line (a sign) → 1 point
- Points of the player above to move are added
- Points of the other player are subtracted

# Games and search – search space

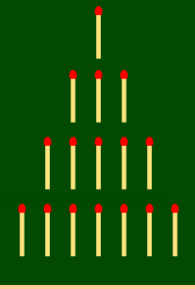


Game's strategies → MiniMax

Example – *Tic-Tac-Toe*



# Games and search – search space



## Game's strategies → MiniMax

### □ Example – NIM game

#### ■ Input :

- N stacks, each of them having  $p_i$  objects. 2 players A and B extract, alternatively, from a single stack any number of objects. The player that performs the last move wins the game. Player A moves first.

#### ■ Output:

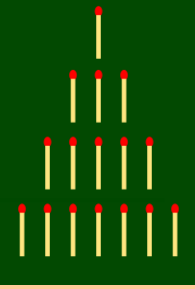
- Determine if player A has a winning strategy and, if it has, program the A's moves, while that of B's are read from keyboard.

#### ■ Study case

- 3 stacks with 1, 1 and 2, respectively, objects



# Games and search – search space



Game's strategies → MiniMax

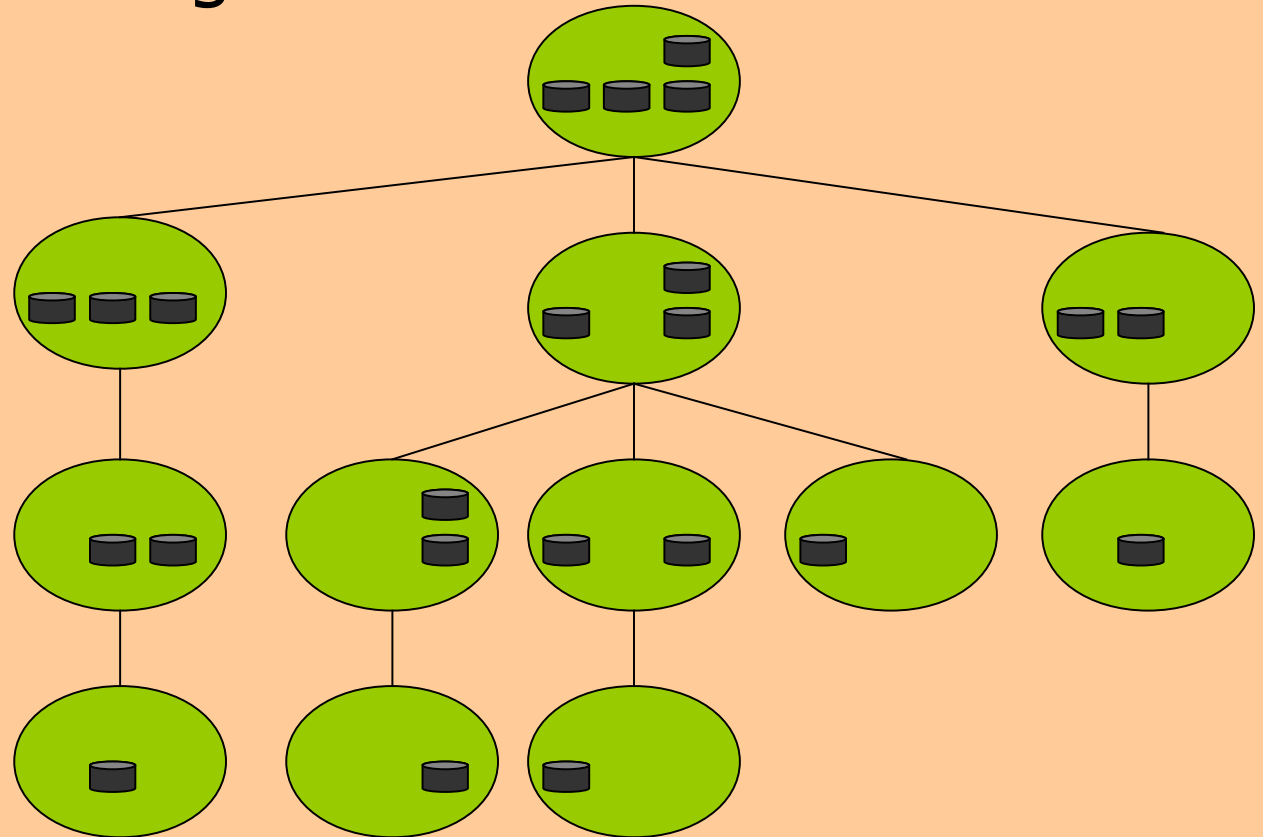
□ Example – NIM game

Max

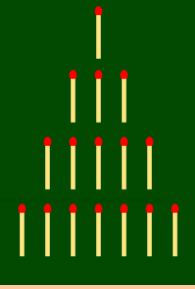
Min

Max

Min



# Games and search – search space



Game's strategies → MiniMax

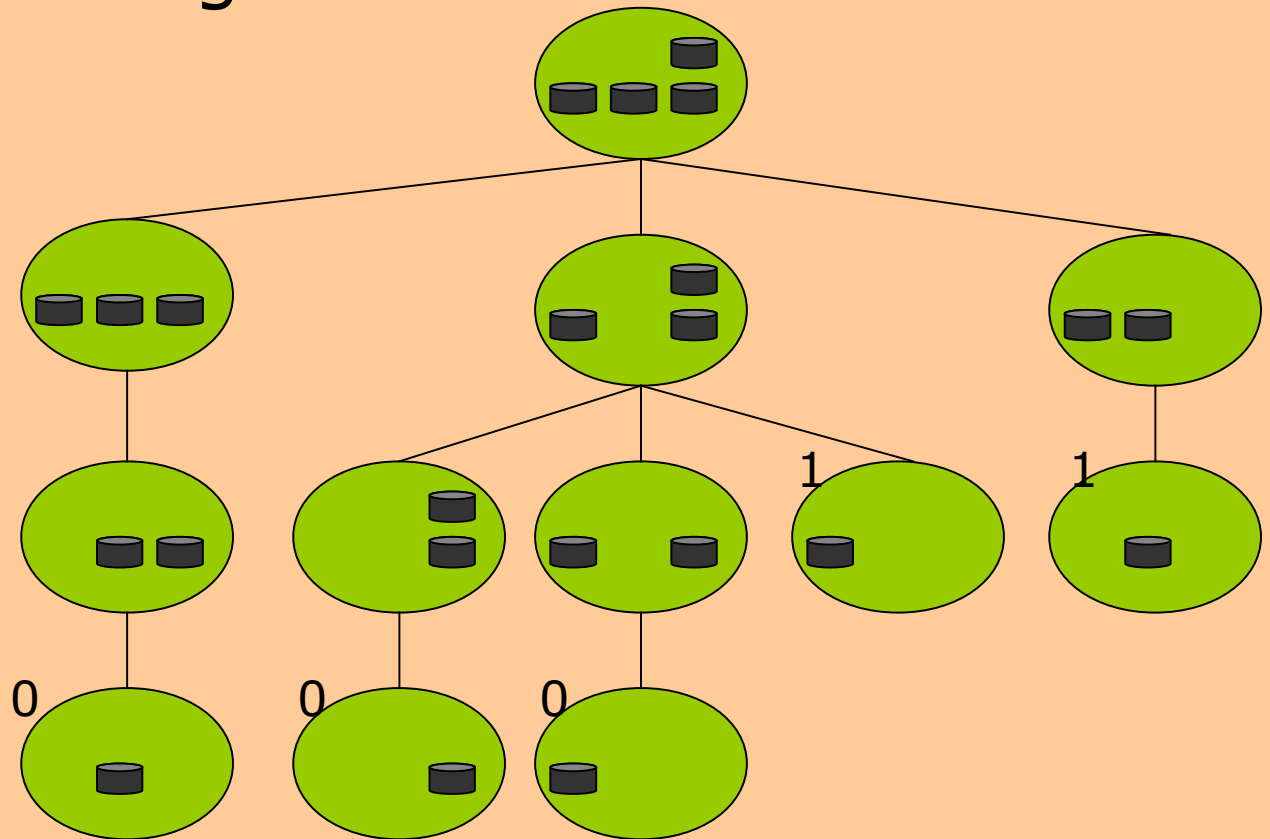
□ Example – NIM game

Max

Min

Max

Min



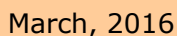


## Example – NIM game

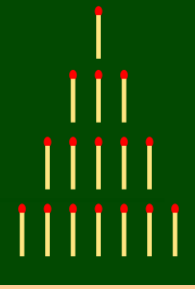
# Min

# Max

Min



# Games and search – search space



Game's strategies → MiniMax

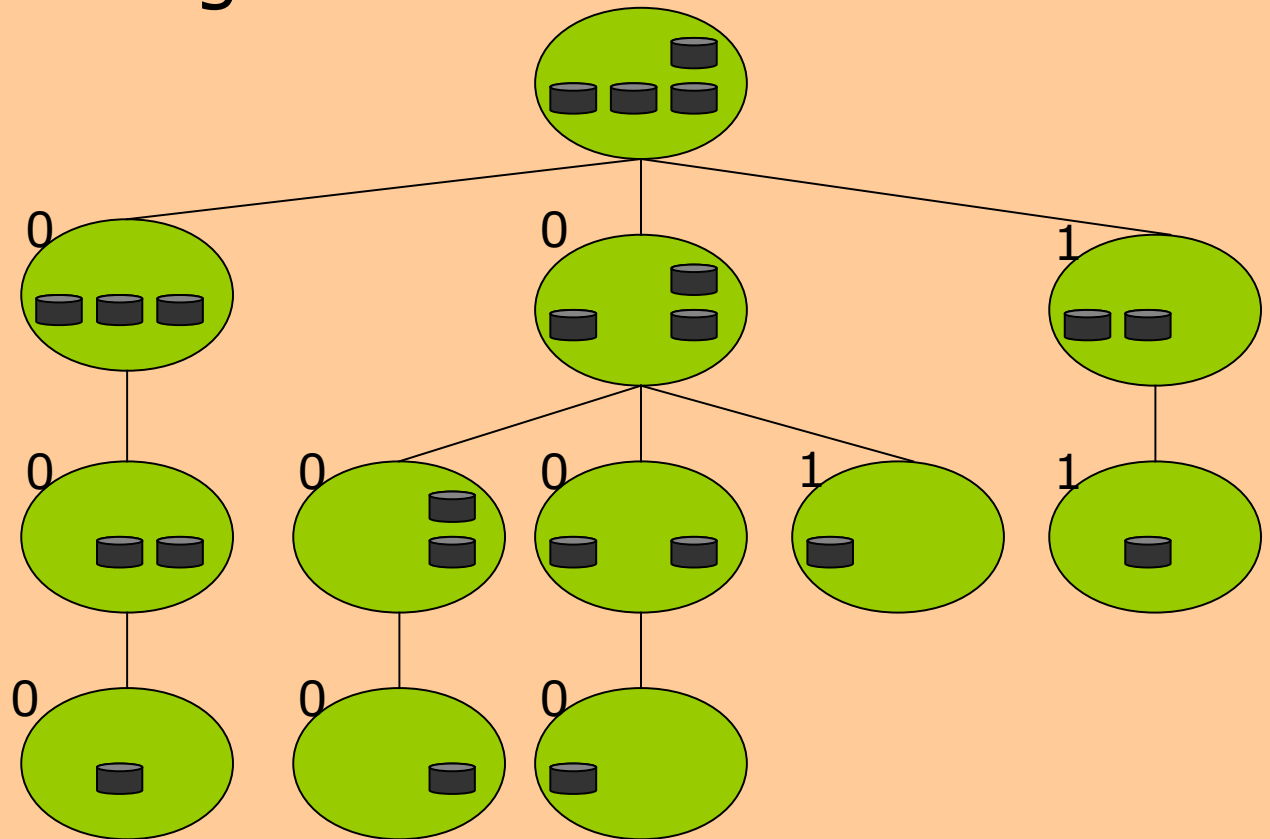
□ Example – NIM game

Max

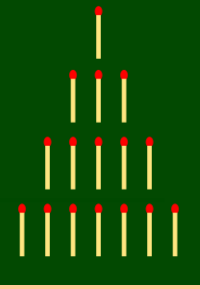
Min

Max

Min



# Games and search – search space



Game's strategies → MiniMax

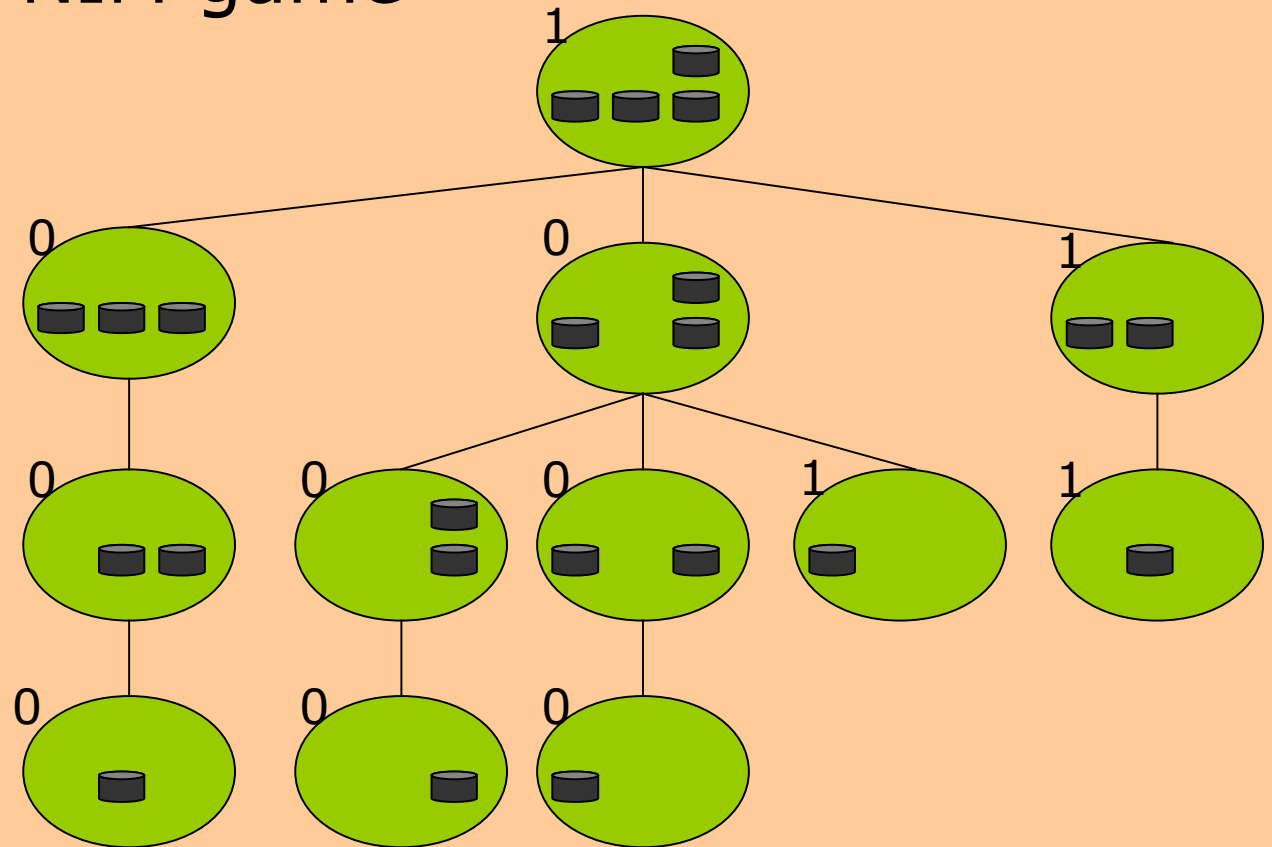
□ Example – NIM game

Max

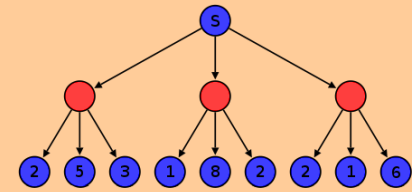
Min

Max

Min



# Games and search – search space

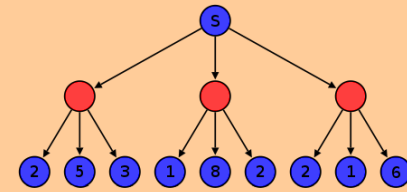


Game's strategies → MiniMax

## □ Difficulties

- Not explore the entire tree
  - Depth-first search
- When the game starts, we have to fix
  - A maximal depth
    - Which is the optimal depth?
      - Large depth → long time to search
      - Small depth → some paths can be missed (early sacrifices for later gains)
  - An evaluation function
    - Each node (state) must be evaluated
    - How? → by using heuristic functions

# Games and search – search space



Game's strategies → MiniMax

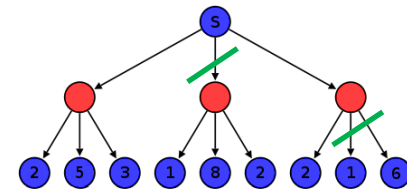
## ❑ Disadvantages

- Only 100 nodes are explored in a second
- Chess – move time = 150s → 150 000 positions  
→ 3-4 move in advance, only

## ❑ Solution

- Avoid some nodes by pruning → alpha-beta  
reducing → MiniMax with  $\alpha$ - $\beta$  pruning

# Games and search – search space



Game's strategies → MiniMax with  $\alpha$ - $\beta$  pruning

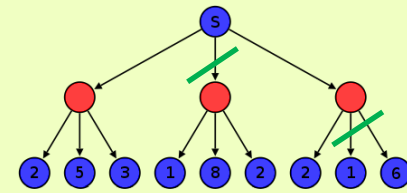
## □ MiniMax

- Creates the entire tree
- Bottom-up propagation of values

## □ MiniMax with $\alpha$ - $\beta$ pruning

- Simultaneously create and propagate
- If a path is bad, no energy is consumed in order to establish how bad is that path
  - Some useless evaluations can be eliminated
  - Some node's expansions can be avoided

# Games and search – search space



## Game's strategies → MiniMax with $\alpha$ - $\beta$ pruning

### □ Theoretical aspects

- Discovered by McCarthy in 1956, but published for first time in a technical report of MIT

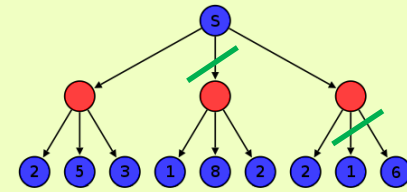
### ■ Main idea

- MiniMax value of the root can be determined by evaluating all the nodes on the searching frontier
- Similarly to branch and bound algorithm

### ■ $\alpha$ - $\beta$ calling

- $\alpha$  - value of the best selection (largest selection) done over the MAX path
  - If a node has a value weaker than  $\alpha$ , then MAX will avoid it and will cut the sub-tree rooted in that node
- $\beta$  - similarly of  $\alpha$ , but for MIN player

# Games and search – search space



## Game's strategies → MiniMax with $\alpha$ - $\beta$ pruning

### □ Theoretical aspects

#### ■ $\alpha$

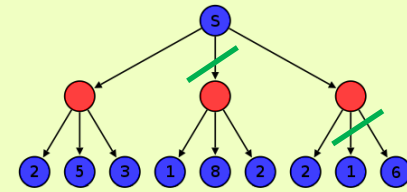
- Value of the best selection (largest selection) performed until now over its path by the MAX player
- Minimum score that can be obtained by MAX player
- If a node  $v$  is weaker (less) than  $\alpha$ , MAX will avoid it by eliminating that branch →
  - If the search is in a MIN node whose value  $\leq \alpha$ , then all children of that node that are not worth exploring because they will be ignored anyway by MAX player

#### ■ $\beta$

- The smallest value found until now over its path by MIN player
- Minimum score that can MAX player hopes to obtain
- If a node  $v$  is better (greater) than  $\beta$ , MIN will avoid it by eliminating that branch →
  - If the search is in a MAX node whose value  $\geq \beta$ , then all children of that node that are not worth exploring because they will be ignored anyway by MIN player



# Games and search – search space

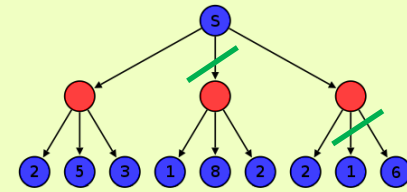


## Game's strategies → MiniMax with $\alpha$ - $\beta$ pruning

### □ Theoretical aspects

- $\alpha$  value of a node
  - Initially, the score of that node (if the node is a leaf) or  $-\infty$
  - Than,
    - MAX node = highest score of children
    - MIN node =  $\alpha$  value of the predecessor
- $\beta$  value of a node
  - Initially, the score of that node (if the node is a leaf) or  $+\infty$
  - Than,
    - MIN node = smallest score of children
    - MAX node =  $\beta$  value of the predecessor
- Score of a node
  - MAX node → final  $\alpha$  value
  - MIN node → final  $\beta$  value

# Games and search – search space

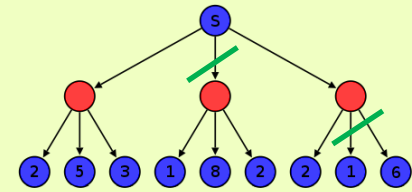


## Game's strategies → MiniMax with $\alpha$ - $\beta$ pruning

### □ Algorithm

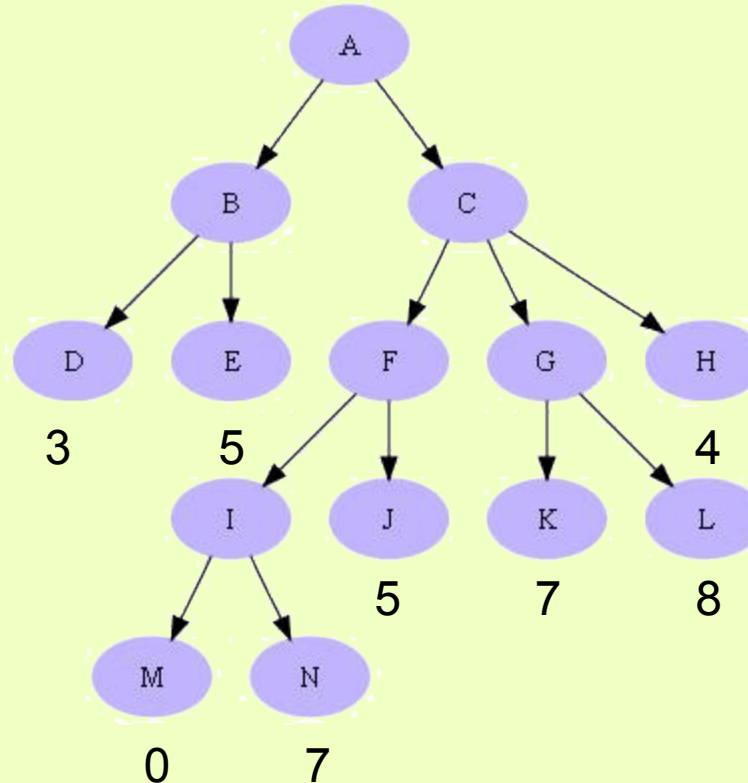
```
int backMiniMaxAB(Node N, int A, int B){
    Set Alpha value of N to A and Beta value of N to B;
    if N is a leaf
        return the estimated score of this leaf
    else{
        if N is a Min node
            for each child Ni of N {
                Val = backMiniMaxAB(Ni, Alpha of N, Beta of N);
                Beta value of N = minim(Beta value of N, Val);
                if Beta value of N ≤ Alpha value of N then exit loop;
            }
        return Beta value of N;
        else // N is a Max node
            for each child Ni of N do {
                Val = MINIMAX-AB(Ni, Alpha of N, Beta of N);
                Alpha value of N = Max{Alpha value of N, Val};
                if Alpha value of N ≥ Beta value of N then exit loop;
            }
        Return Alpha value of N;
    }
}
```

# Games and search – search space

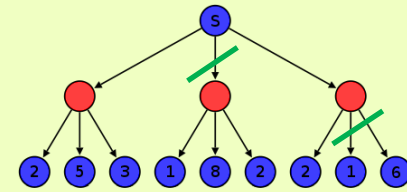


Game's strategies  $\rightarrow$  MiniMax with  $\alpha$ - $\beta$  pruning

□ Example



# Games and search – search space



Game's strategies → MiniMax with  $\alpha$ - $\beta$  pruning

□ Example

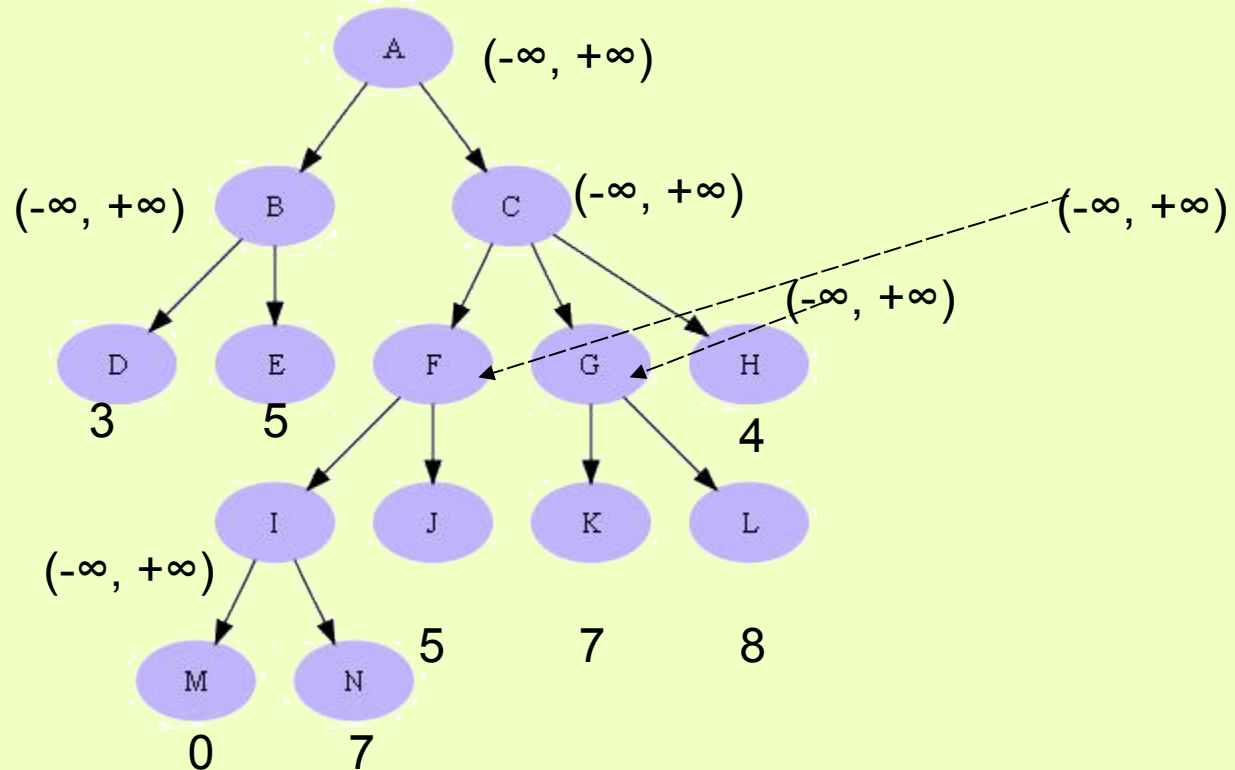
MAX

MIN

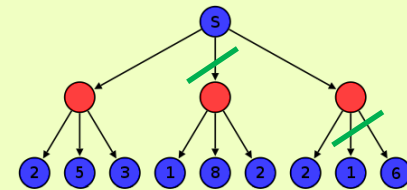
MAX

MIN

MAX



# Games and search – search space



Game's strategies  $\rightarrow$  MiniMax with  $\alpha$ - $\beta$  pruning

□ Example

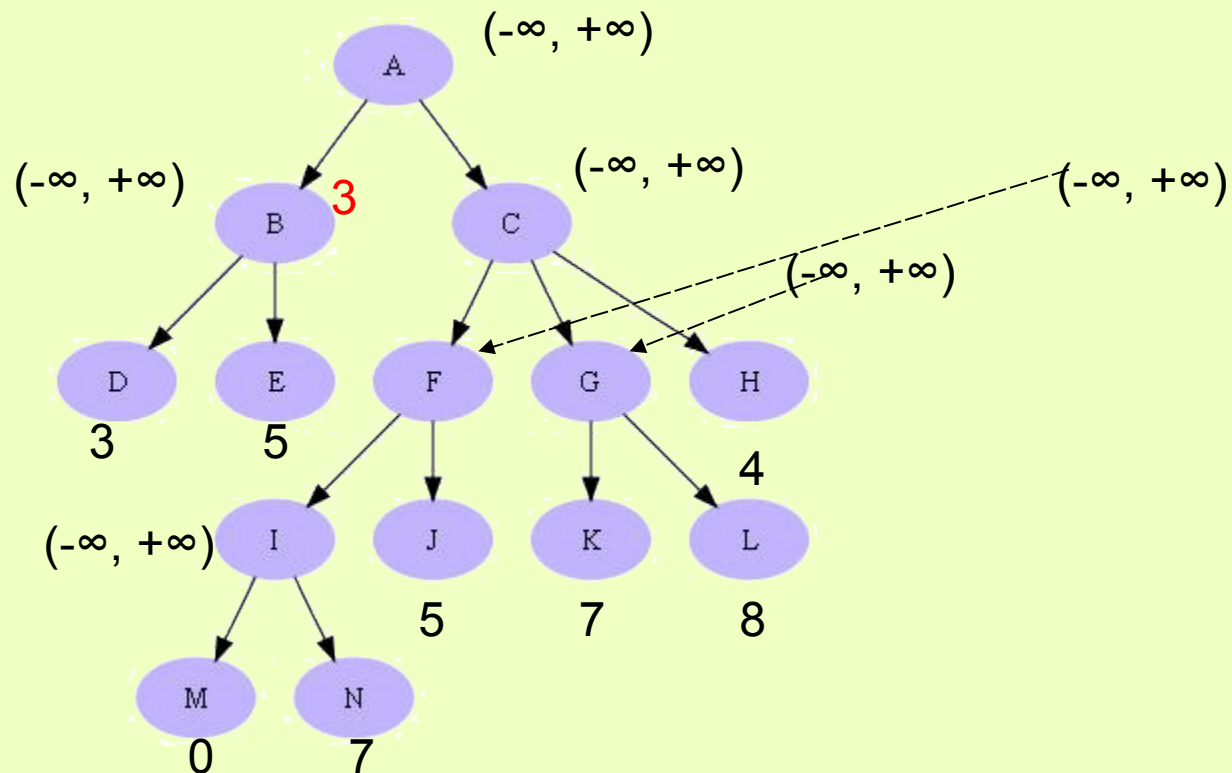
MAX( $\alpha$ )

MIN( $\beta$ )

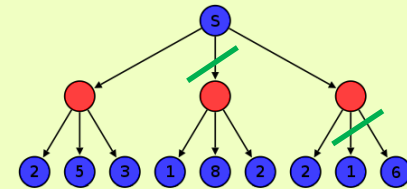
MAX( $\alpha$ )

MIN( $\beta$ )

MAX( $\alpha$ )



# Games and search – search space



Game's strategies  $\rightarrow$  MiniMax with  $\alpha$ - $\beta$  pruning

- Example

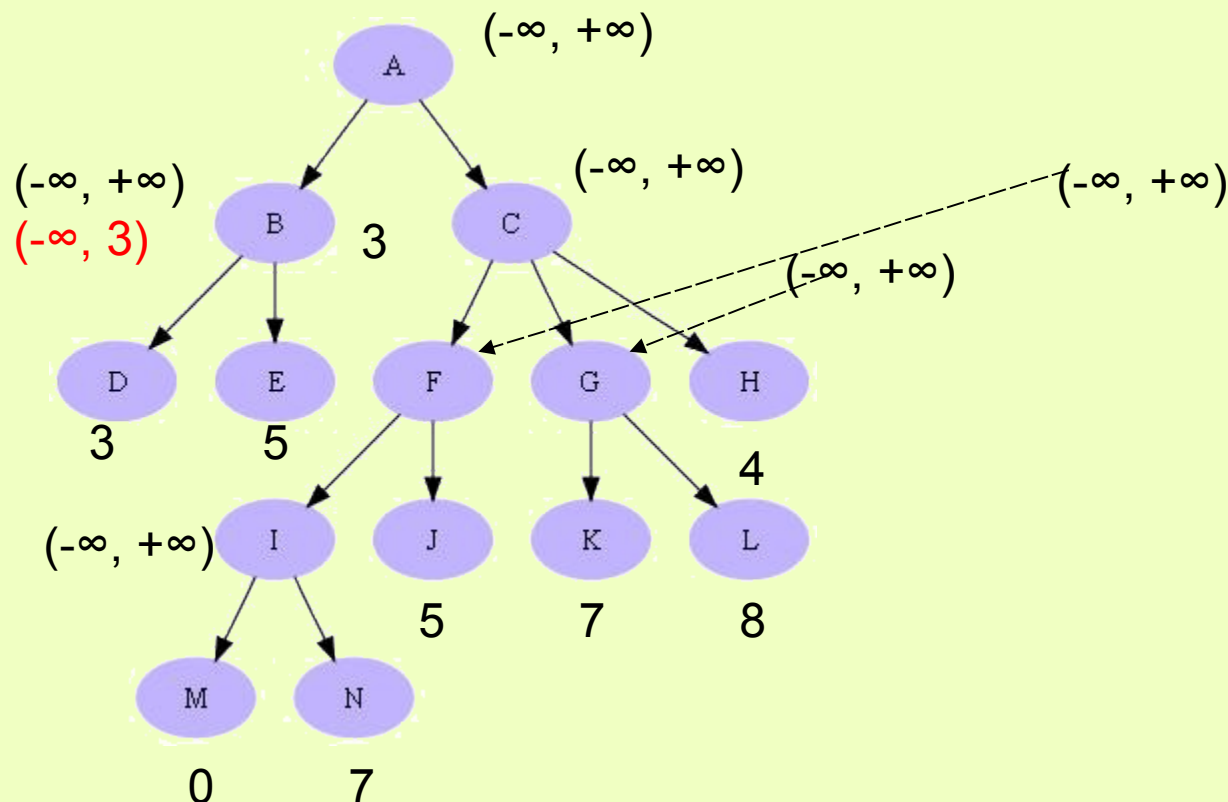
MAX( $\alpha$ )

MIN( $\beta$ )

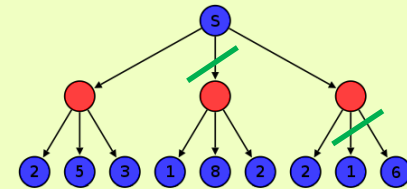
MAX( $\alpha$ )

MIN( $\beta$ )

MAX( $\alpha$ )



# Games and search – search space



Game's strategies → MiniMax with  $\alpha$ - $\beta$  pruning

□ Example

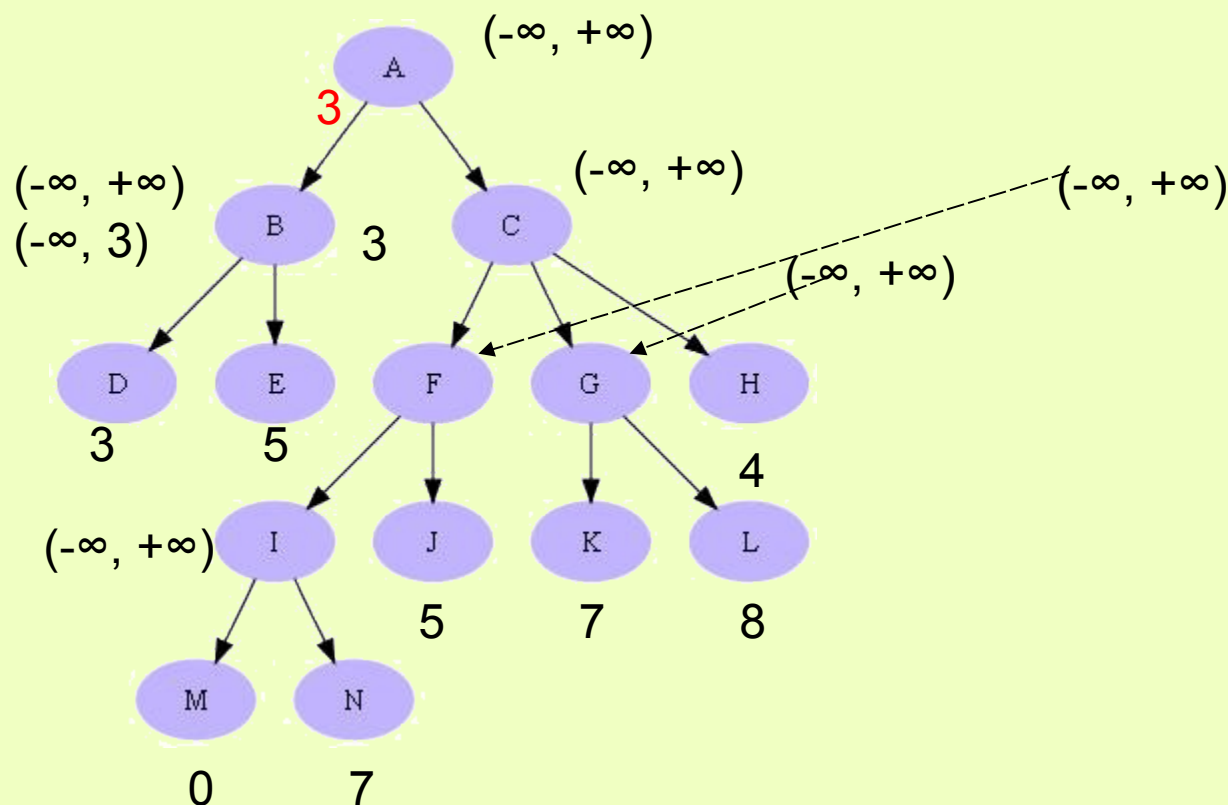
MAX( $\alpha$ )

MIN( $\beta$ )

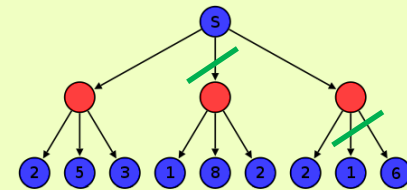
MAX( $\alpha$ )

MIN( $\beta$ )

MAX( $\alpha$ )



# Games and search – search space



Game's strategies → MiniMax with  $\alpha$ - $\beta$  pruning

□ Example

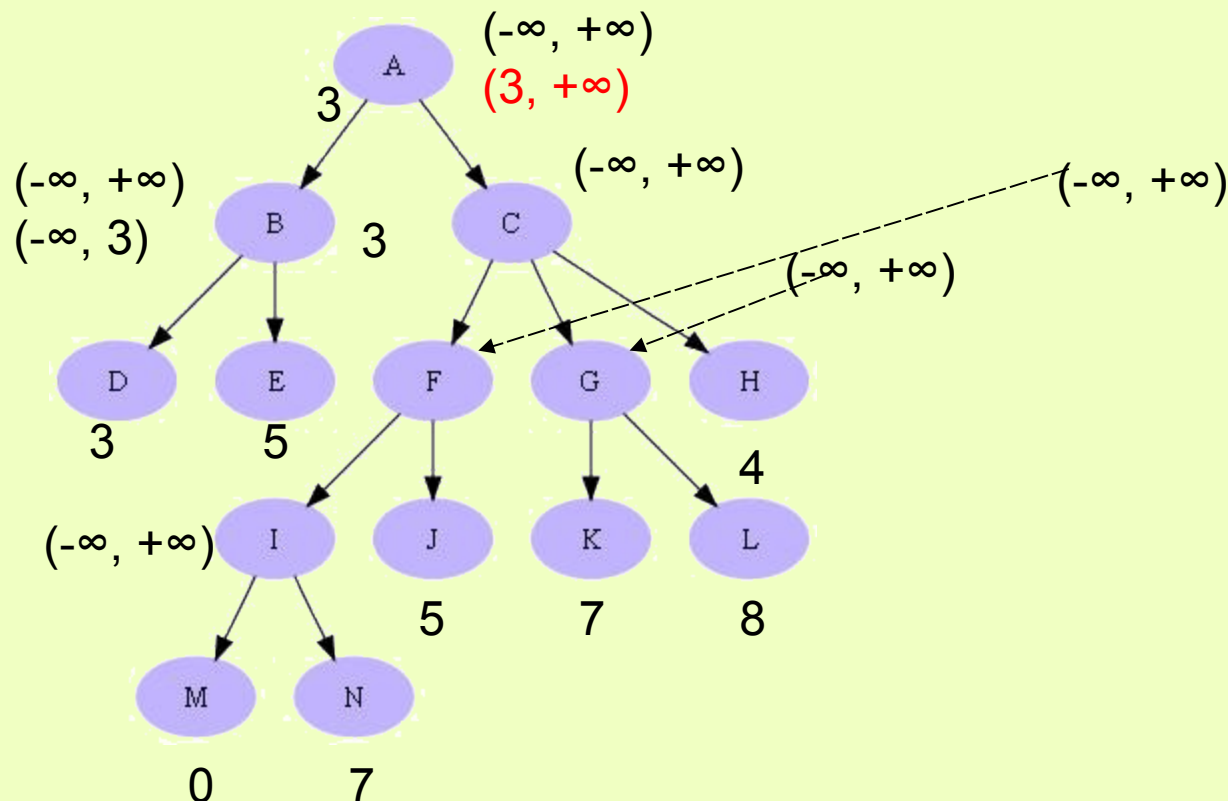
MAX( $\alpha$ )

MIN( $\beta$ )

MAX( $\alpha$ )

MIN( $\beta$ )

MAX( $\alpha$ )





## Example

The diagram shows a search tree for a minimax problem. The root node is A, with children B and C. B has children D and E. C has children F, G, and H. F has children I and J. G has children K and L. I has children M and N. Leaf nodes have values: D=3, E=5, J=5, K=7, L=8. Internal nodes have values: A=3, B=3, C=3 (highlighted in red), I=3. Pruning is indicated by dashed lines from C to G and H, labeled with  $(-\infty, +\infty)$ .

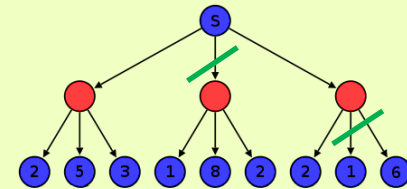
# 07 Artificial Intelligence

## Artificial Intelligence - adversarial search

73

MAX(a)

# Games and search – search space



Game's strategies → MiniMax with  $\alpha$ - $\beta$  pruning

□ Example

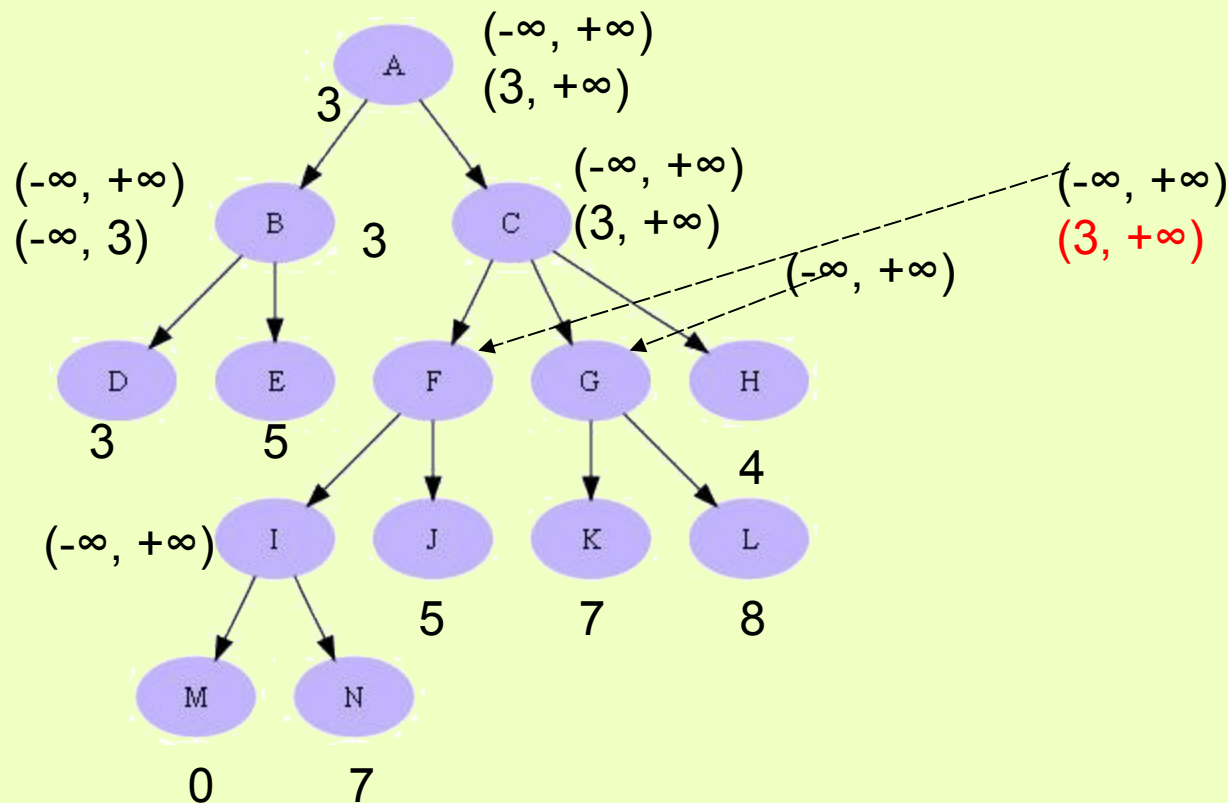
MAX( $\alpha$ )

MIN( $\beta$ )

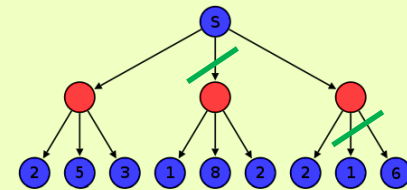
MAX( $\alpha$ )

MIN( $\beta$ )

MAX( $\alpha$ )



# Games and search – search space



Game's strategies → MiniMax with  $\alpha$ - $\beta$  pruning

□ Example

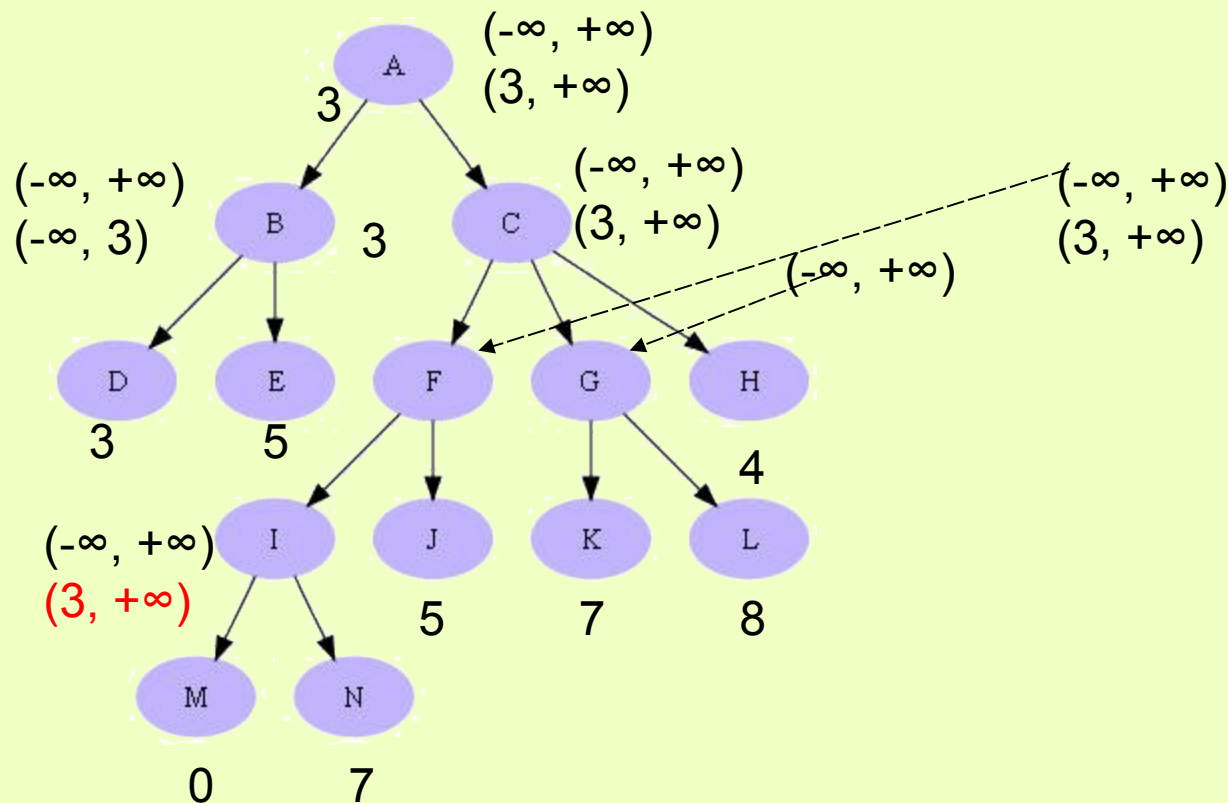
MAX( $\alpha$ )

MIN( $\beta$ )

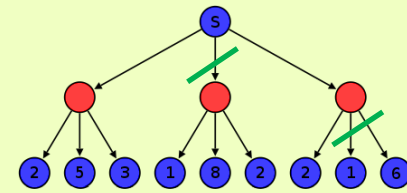
MAX( $\alpha$ )

MIN( $\beta$ )

MAX( $\alpha$ )



# Games and search – search space



Game's strategies → MiniMax with  $\alpha$ - $\beta$  pruning

□ Example

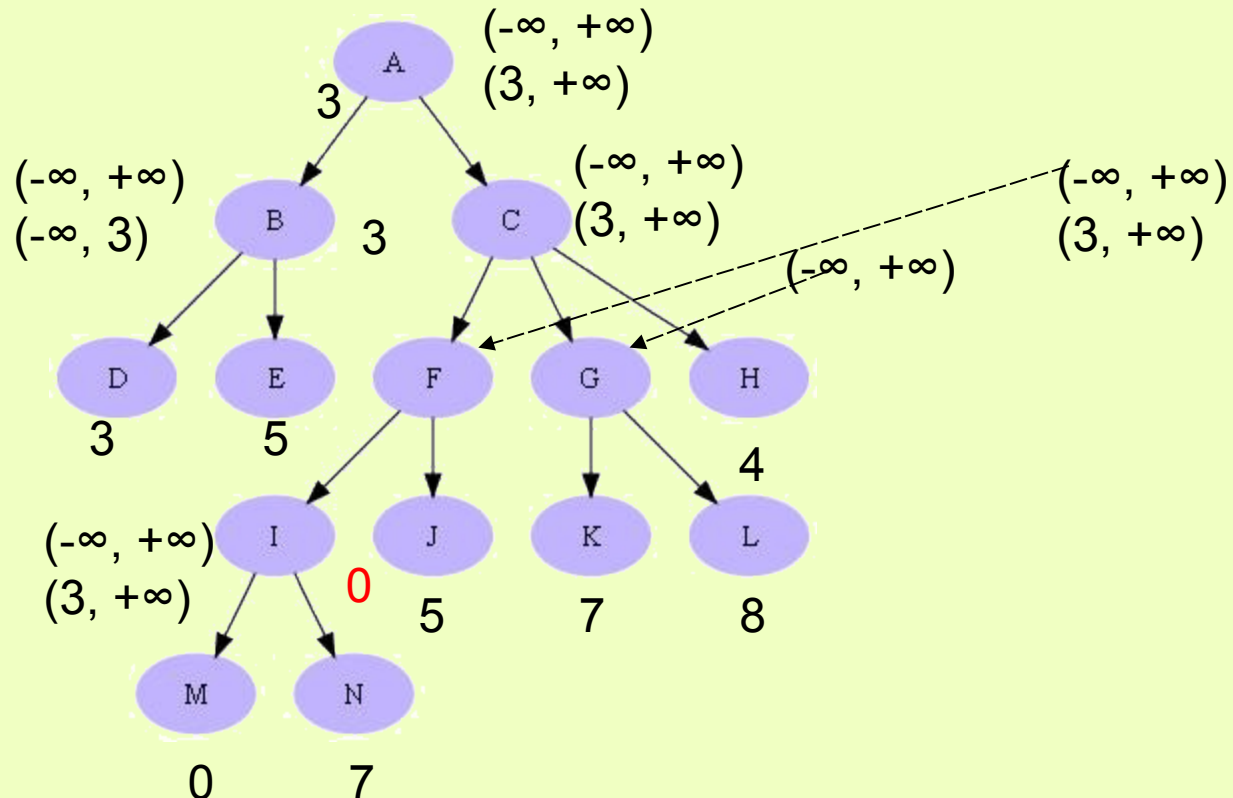
MAX( $\alpha$ )

MIN( $\beta$ )

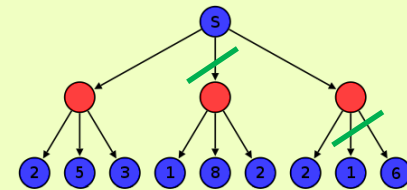
MAX( $\alpha$ )

MIN( $\beta$ )

MAX( $\alpha$ )



# Games and search – search space



Game's strategies  $\rightarrow$  MiniMax with  $\alpha$ - $\beta$  pruning

□ Example

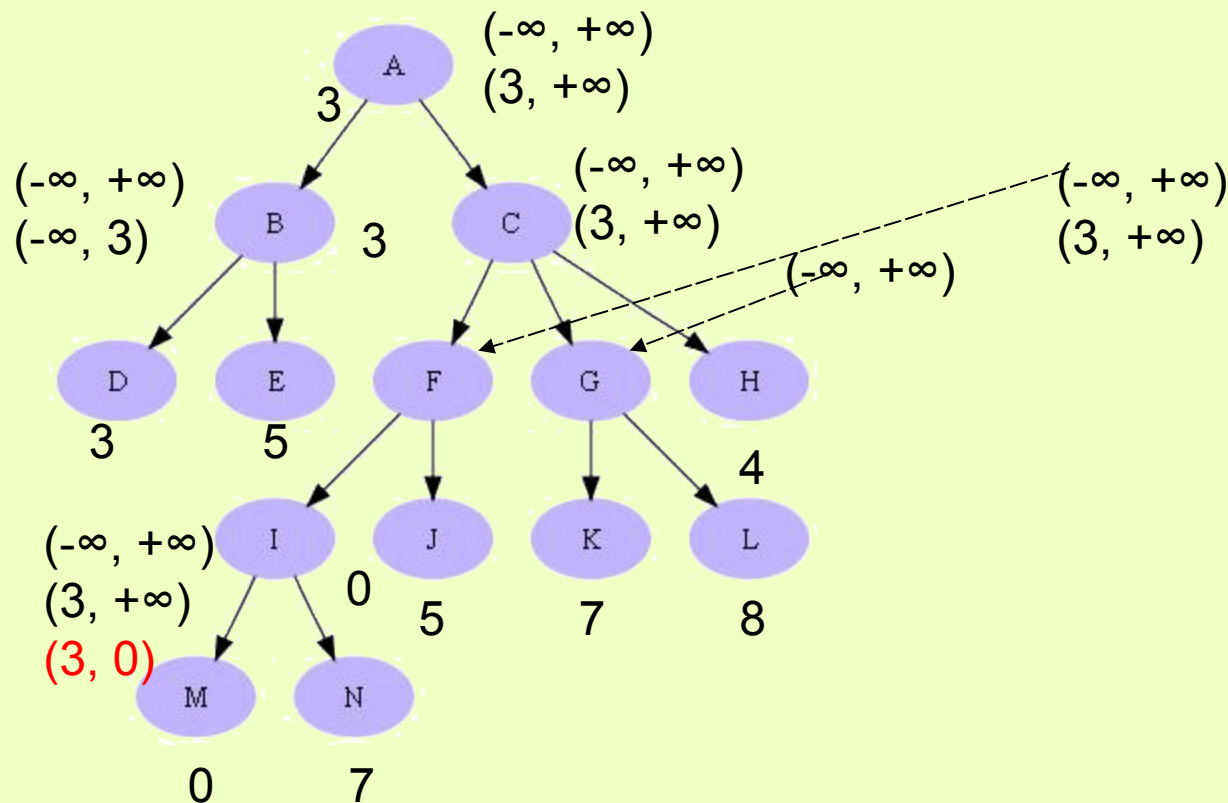
MAX( $\alpha$ )

MIN( $\beta$ )

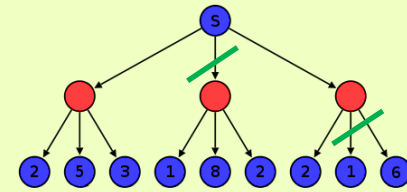
MAX( $\alpha$ )

MIN( $\beta$ )

MAX( $\alpha$ )



# Games and search – search space



Game's strategies → MiniMax with  $\alpha$ - $\beta$  pruning

□ Example

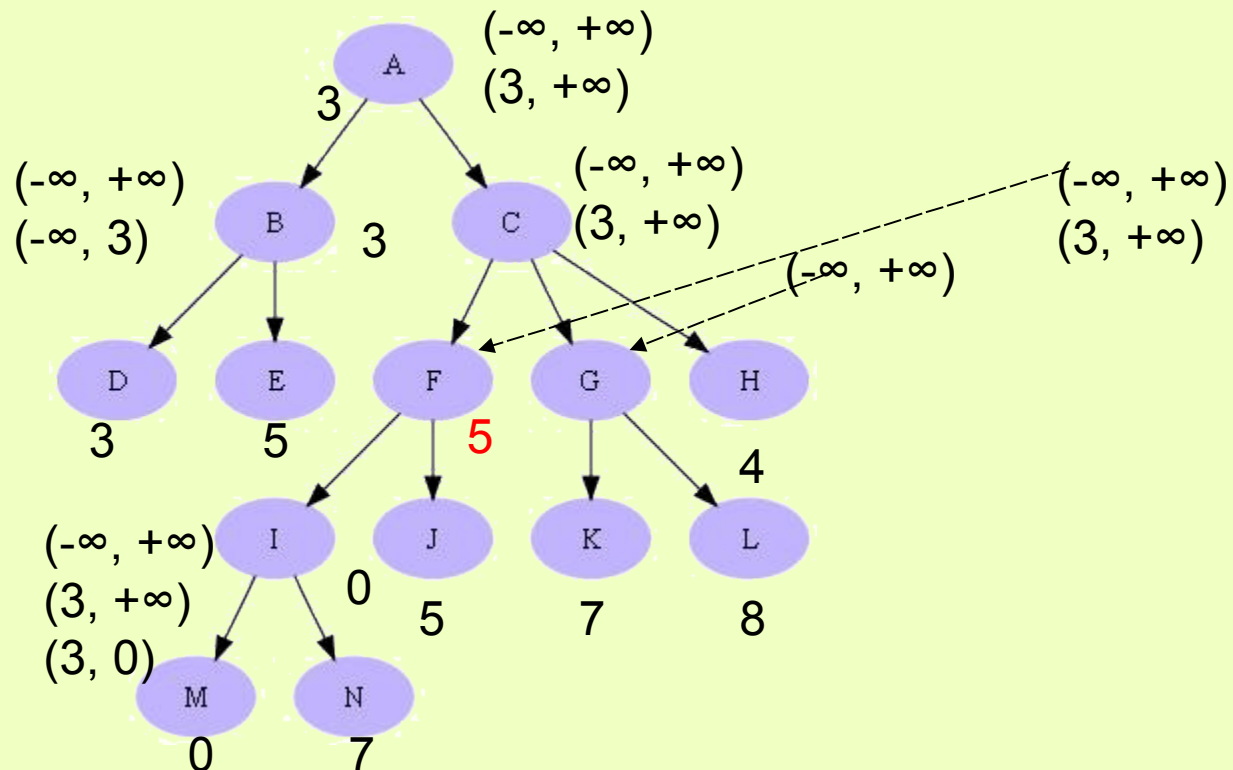
MAX( $\alpha$ )

MIN( $\beta$ )

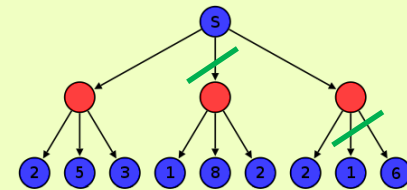
MAX( $\alpha$ )

MIN( $\beta$ )

MAX( $\alpha$ )



# Games and search – search space



Game's strategies → MiniMax with  $\alpha$ - $\beta$  pruning

□ Example

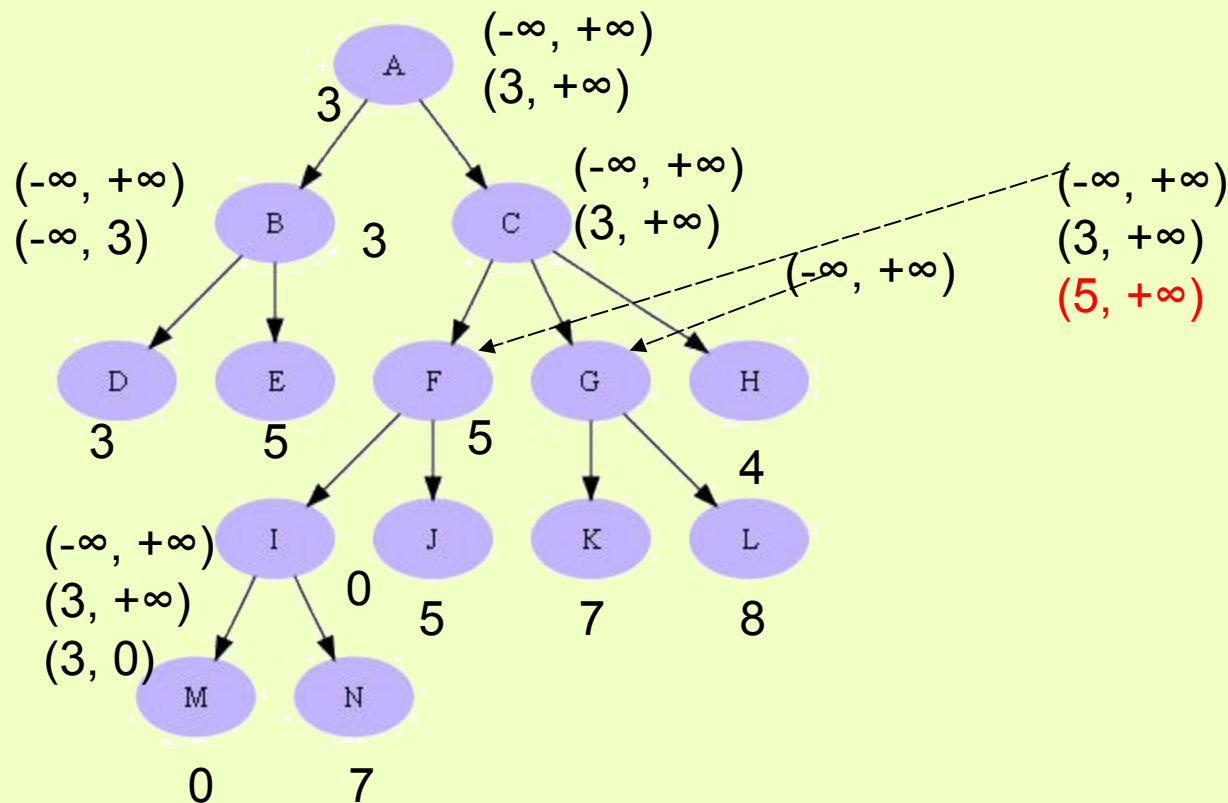
MAX( $\alpha$ )

MIN( $\beta$ )

MAX( $\alpha$ )

MIN( $\beta$ )

MAX( $\alpha$ )



## Example

The diagram shows a search tree with the following structure and values:

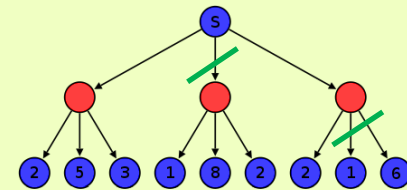
- Node A: Value 3, Range  $(-\infty, +\infty)$  and  $(3, +\infty)$
- Node B: Value 3, Range  $(-\infty, +\infty)$  and  $(-\infty, 3)$
- Node C: Value 5 (highlighted in red), Range  $(-\infty, +\infty)$  and  $(3, +\infty)$
- Node D: Value 3, Range  $(-\infty, +\infty)$
- Node E: Value 5, Range  $(-\infty, +\infty)$
- Node F: Value 5, Range  $(-\infty, +\infty)$
- Node G: Value 4, Range  $(-\infty, +\infty)$
- Node H: Value 8, Range  $(-\infty, +\infty)$
- Node I: Value 0, Range  $(-\infty, +\infty)$  and  $(3, +\infty)$
- Node J: Value 5, Range  $(-\infty, +\infty)$
- Node K: Value 7, Range  $(-\infty, +\infty)$
- Node L: Value 8, Range  $(-\infty, +\infty)$
- Node M: Value 3, Range  $(-\infty, +\infty)$
- Node N: Value 0, Range  $(-\infty, +\infty)$

Dashed arrows indicate the backpropagation of values from the leaf nodes to their parents.

MAX(a)



# Games and search – search space



Game's strategies → MiniMax with  $\alpha$ - $\beta$  pruning

□ Example

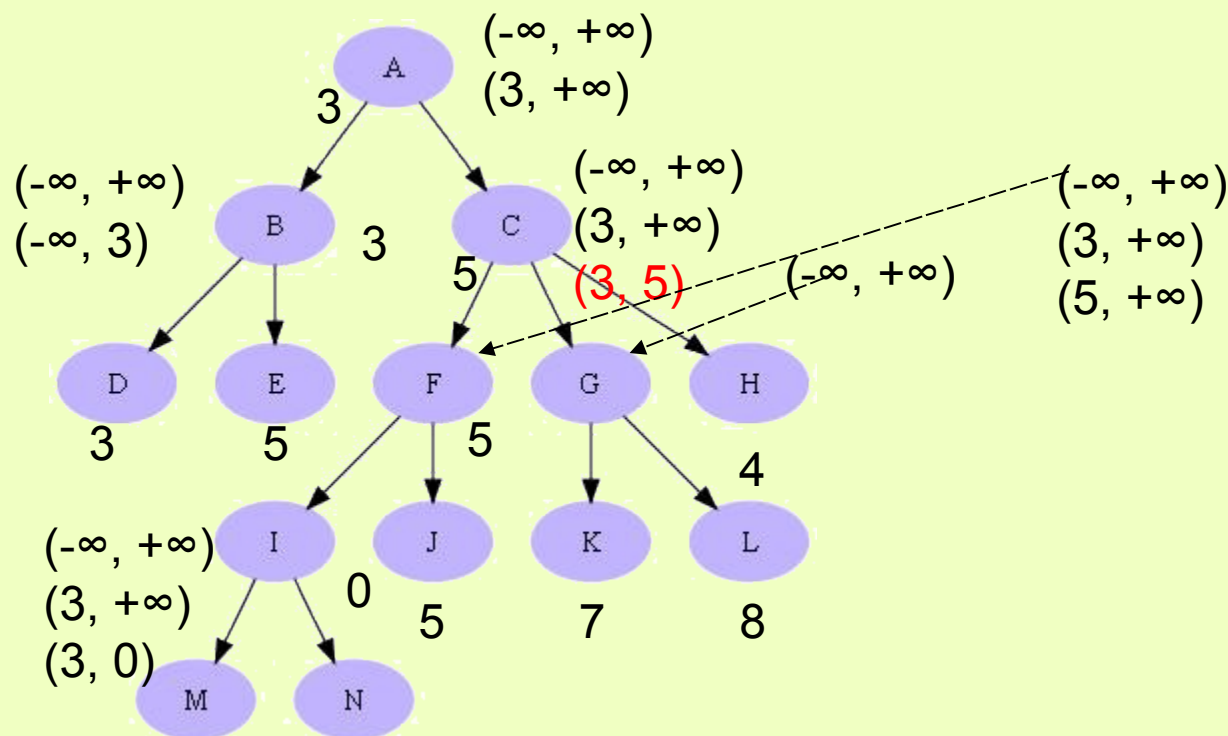
MAX( $\alpha$ )

MIN( $\beta$ )

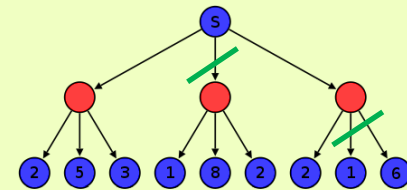
MAX( $\alpha$ )

MIN( $\beta$ )

MAX( $\alpha$ )



# Games and search – search space



Game's strategies → MiniMax with  $\alpha$ - $\beta$  pruning

□ Example

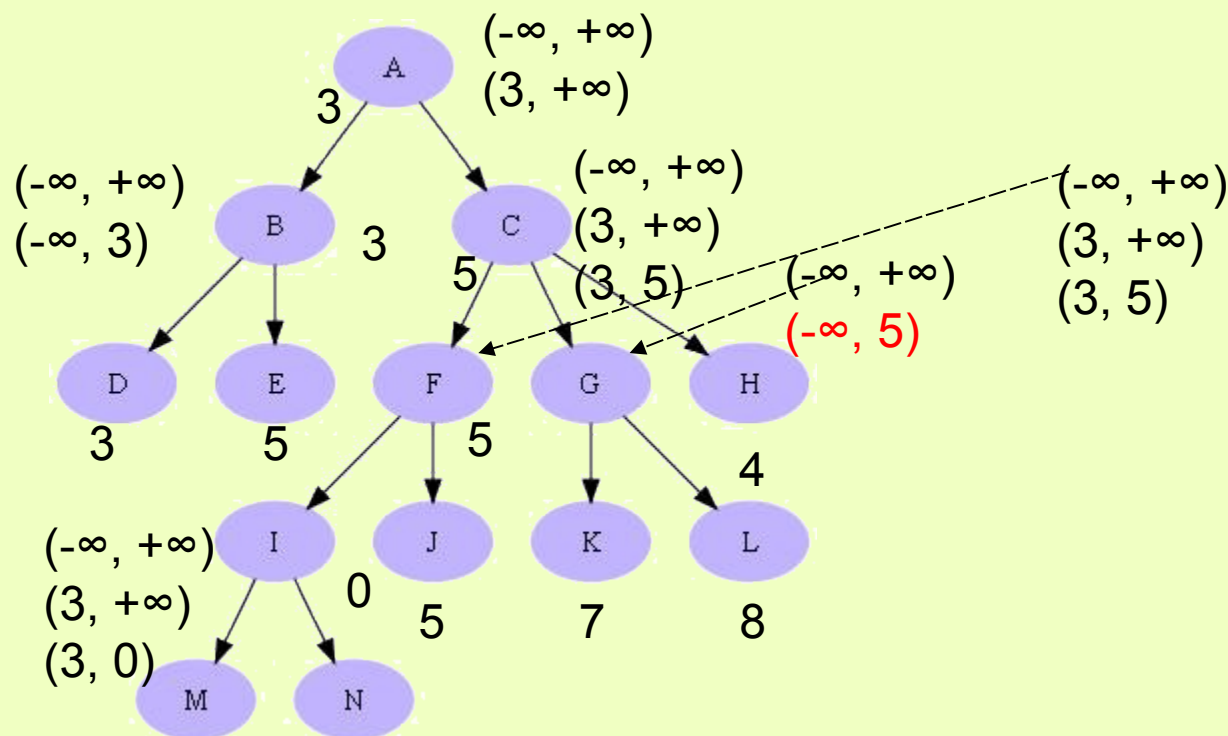
MAX( $\alpha$ )

MIN( $\beta$ )

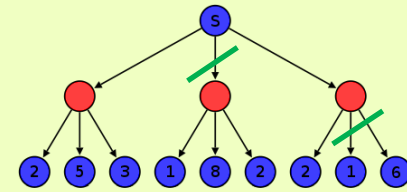
MAX( $\alpha$ )

MIN( $\beta$ )

MAX( $\alpha$ )



# Games and search – search space



Game's strategies → MiniMax with  $\alpha$ - $\beta$  pruning

□ Example

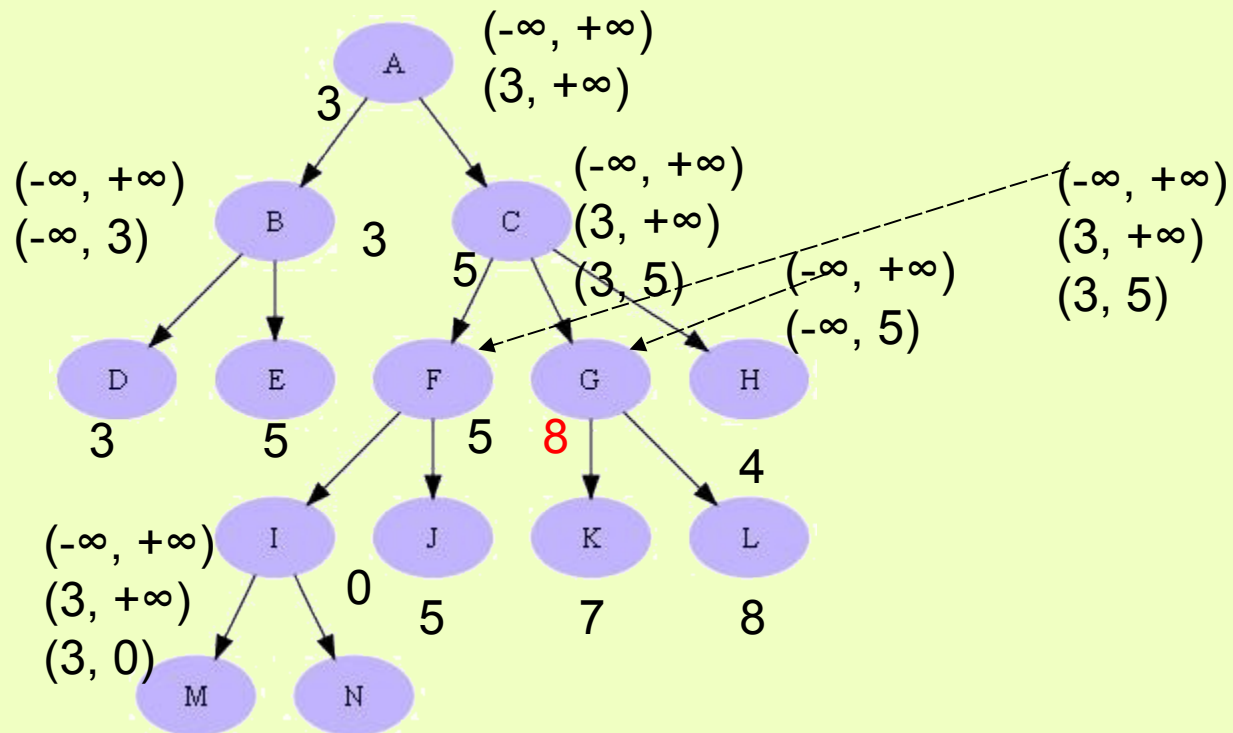
MAX( $\alpha$ )

MIN( $\beta$ )

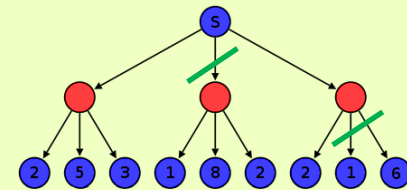
MAX( $\alpha$ )

MIN( $\beta$ )

MAX( $\alpha$ )



# Games and search – search space



Game's strategies → MiniMax with  $\alpha$ - $\beta$  pruning

□ Example

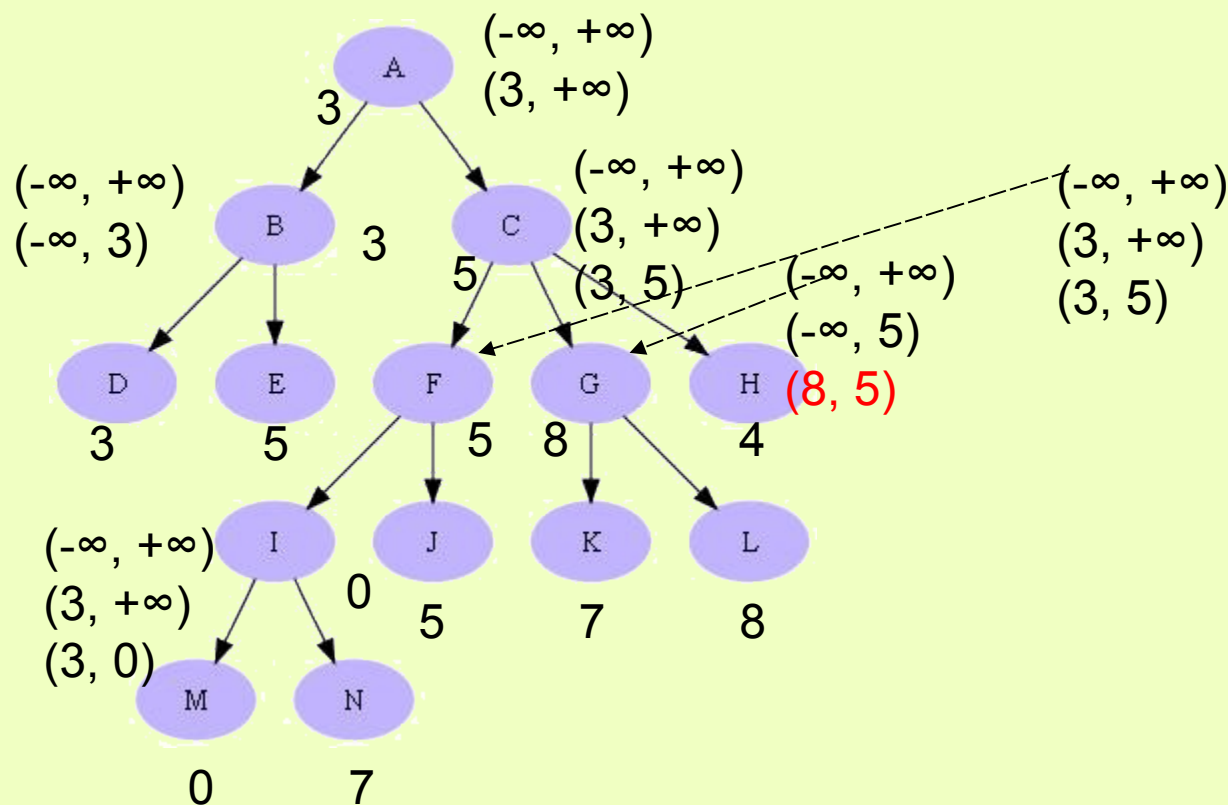
MAX( $\alpha$ )

MIN( $\beta$ )

MAX( $\alpha$ )

MIN( $\beta$ )

MAX( $\alpha$ )



## Example

A search tree diagram illustrating the A\* algorithm. The root node is A, with children B and C. B has children D and E. C has children F, G, and H. F has children I and J. G has children K and L. I has children M and N. Each node is labeled with its key value (in a purple oval) and its f-cost (in a black box). The f-cost for node A is  $(-\infty, +\infty)$  and  $(3, +\infty)$ . The f-cost for node B is  $(-\infty, +\infty)$  and  $(-\infty, 3)$ . The f-cost for node C is  $(-\infty, +\infty)$  and  $(3, +\infty)$ . The f-cost for node D is  $(-\infty, +\infty)$  and  $(-\infty, 3)$ . The f-cost for node E is  $(-\infty, +\infty)$  and  $(-\infty, 5)$ . The f-cost for node F is  $(-\infty, +\infty)$  and  $(5, 5)$ . The f-cost for node G is  $(-\infty, +\infty)$  and  $(8, 5)$ . The f-cost for node H is  $(-\infty, +\infty)$  and  $(3, 5)$ . The f-cost for node I is  $(-\infty, +\infty)$  and  $(3, 0)$ . The f-cost for node J is  $(-\infty, +\infty)$  and  $(5, 5)$ . The f-cost for node K is  $(-\infty, +\infty)$  and  $(7, 5)$ . The f-cost for node L is  $(-\infty, +\infty)$  and  $(8, 5)$ . The f-cost for node M is  $(-\infty, +\infty)$  and  $(3, 0)$ . The f-cost for node N is  $(-\infty, +\infty)$  and  $(0, 5)$ . The tree shows the search path from A to B to C to F to I to M, with the final path highlighted in red.

Diagram illustrating a graph structure with nodes A through M. The nodes are represented by purple ovals, and the edges are directed arrows. The graph is divided into four regions, each associated with a set of intervals (shown in boxes):

- Region 1 (Top Left):** Contains nodes B and C. The intervals are  $(-\infty, +\infty)$  and  $(-\infty, 3)$ .
- Region 2 (Top Right):** Contains nodes C and G. The intervals are  $(3, 5)$  and  $(-\infty, +\infty)$ .
- Region 3 (Bottom Left):** Contains nodes I and M. The intervals are  $(-\infty, +\infty)$ ,  $(3, +\infty)$ , and  $(3, 0)$ .
- Region 4 (Bottom Right):** Contains nodes G and H. The intervals are  $(-\infty, +\infty)$ ,  $(-\infty, 5)$ , and  $(8, 5)$ .

The nodes and their associated values are:

- A: 3
- B: 3
- C: 5 (with a red 4 next to it)
- D: 3
- E: 5
- F: 5
- G: 8
- H: 4
- I: 0
- J: 5
- K: 7
- L: 8
- M: 0
- N: 5

The edges and their weights are:

- A → B (weight 3)
- A → C (weight 5)
- B → D (weight 3)
- B → E (weight 5)
- C → F (weight 5)
- C → G (weight 8)
- C → H (weight 4)
- F → I (weight 0)
- F → J (weight 5)
- G → K (weight 7)
- G → L (weight 8)
- I → M (weight 0)
- I → N (weight 5)

Dashed arrows indicate additional connections from node C to nodes G and H.

Diagram illustrating a graph structure with nodes A through M. The nodes are represented by purple ovals, and the edges are directed arrows. The graph is divided into four regions, each associated with a set of intervals (shown in boxes):

- Region 1 (Top Left):** Contains nodes B and C. The intervals are  $(-\infty, +\infty)$  and  $(-\infty, 3)$ .
- Region 2 (Top Right):** Contains nodes C and G. The intervals are  $(3, 5)$  and  $(-\infty, +\infty)$ .
- Region 3 (Bottom Left):** Contains nodes I and M. The intervals are  $(-\infty, +\infty)$ ,  $(3, +\infty)$ , and  $(3, 0)$ .
- Region 4 (Bottom Right):** Contains nodes G and H. The intervals are  $(-\infty, +\infty)$ ,  $(-\infty, 5)$ , and  $(8, 5)$ .

The nodes and their associated values are:

- A: 3
- B: 3
- C: 5 (with a red 4 next to it)
- D: 3
- E: 5
- F: 5
- G: 8
- H: 4
- I: 0
- J: 5
- K: 7
- L: 8
- M: 0
- N: 5

The edges and their weights are:

- A → B (weight 3)
- A → C (weight 5)
- B → D (weight 3)
- B → E (weight 5)
- C → F (weight 5)
- C → G (weight 8)
- C → H (weight 4)
- F → I (weight 0)
- F → J (weight 5)
- G → K (weight 7)
- G → L (weight 8)
- I → M (weight 0)
- I → N (weight 5)

Dashed arrows indicate additional connections from node C to nodes G and H.

Diagram illustrating a graph structure with nodes A through M. The nodes are represented by purple ovals, and the edges are directed arrows. The graph is divided into four regions, each associated with a set of intervals (shown in boxes):

- Region 1 (Top Left):** Contains nodes B and C. The intervals are  $(-\infty, +\infty)$  and  $(-\infty, 3)$ .
- Region 2 (Top Right):** Contains nodes C and G. The intervals are  $(3, 5)$  and  $(-\infty, +\infty)$ .
- Region 3 (Bottom Left):** Contains nodes I and M. The intervals are  $(-\infty, +\infty)$ ,  $(3, +\infty)$ , and  $(3, 0)$ .
- Region 4 (Bottom Right):** Contains nodes G and H. The intervals are  $(-\infty, +\infty)$ ,  $(-\infty, 5)$ , and  $(8, 5)$ .

The nodes and their associated values are:

- A: 3
- B: 3
- C: 5 (with a red 4 next to it)
- D: 3
- E: 5
- F: 5
- G: 8
- H: 4
- I: 0
- J: 5
- K: 7
- L: 8
- M: 0
- N: 5

The edges and their weights are:

- A → B (weight 3)
- A → C (weight 5)
- B → D (weight 3)
- B → E (weight 5)
- C → F (weight 5)
- C → G (weight 8)
- C → H (weight 4)
- F → I (weight 0)
- F → J (weight 5)
- G → K (weight 7)
- G → L (weight 8)
- I → M (weight 0)
- I → N (weight 5)

Dashed arrows indicate additional connections from node C to nodes G and H.

Diagram illustrating a graph structure with nodes A through M. The nodes are represented by purple ovals, and the edges are directed arrows. The graph is divided into four regions, each associated with a set of intervals (shown in boxes):

- Region 1 (Top Left):** Contains nodes B and C. The intervals are  $(-\infty, +\infty)$  and  $(-\infty, 3)$ .
- Region 2 (Top Right):** Contains nodes C and G. The intervals are  $(3, 5)$  and  $(-\infty, +\infty)$ .
- Region 3 (Bottom Left):** Contains nodes I and M. The intervals are  $(-\infty, +\infty)$ ,  $(3, +\infty)$ , and  $(3, 0)$ .
- Region 4 (Bottom Right):** Contains nodes G and H. The intervals are  $(-\infty, +\infty)$ ,  $(-\infty, 5)$ , and  $(8, 5)$ .

The nodes and their associated values are:

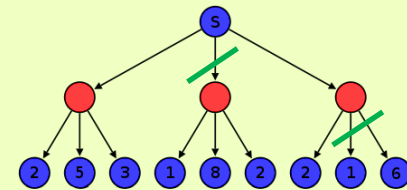
- A: 3
- B: 3
- C: 5 (with a red 4 next to it)
- D: 3
- E: 5
- F: 5
- G: 8
- H: 4
- I: 0
- J: 5
- K: 7
- L: 8
- M: 0
- N: 5

The edges and their weights are:

- A → B (weight 3)
- A → C (weight 5)
- B → D (weight 3)
- B → E (weight 5)
- C → F (weight 5)
- C → G (weight 8)
- C → H (weight 4)
- F → I (weight 0)
- F → J (weight 5)
- G → K (weight 7)
- G → L (weight 8)
- I → M (weight 0)
- I → N (weight 5)

Dashed arrows indicate additional connections from node C to nodes G and H.

# Games and search – search space



Game's strategies → MiniMax with  $\alpha$ - $\beta$  pruning

□ Example

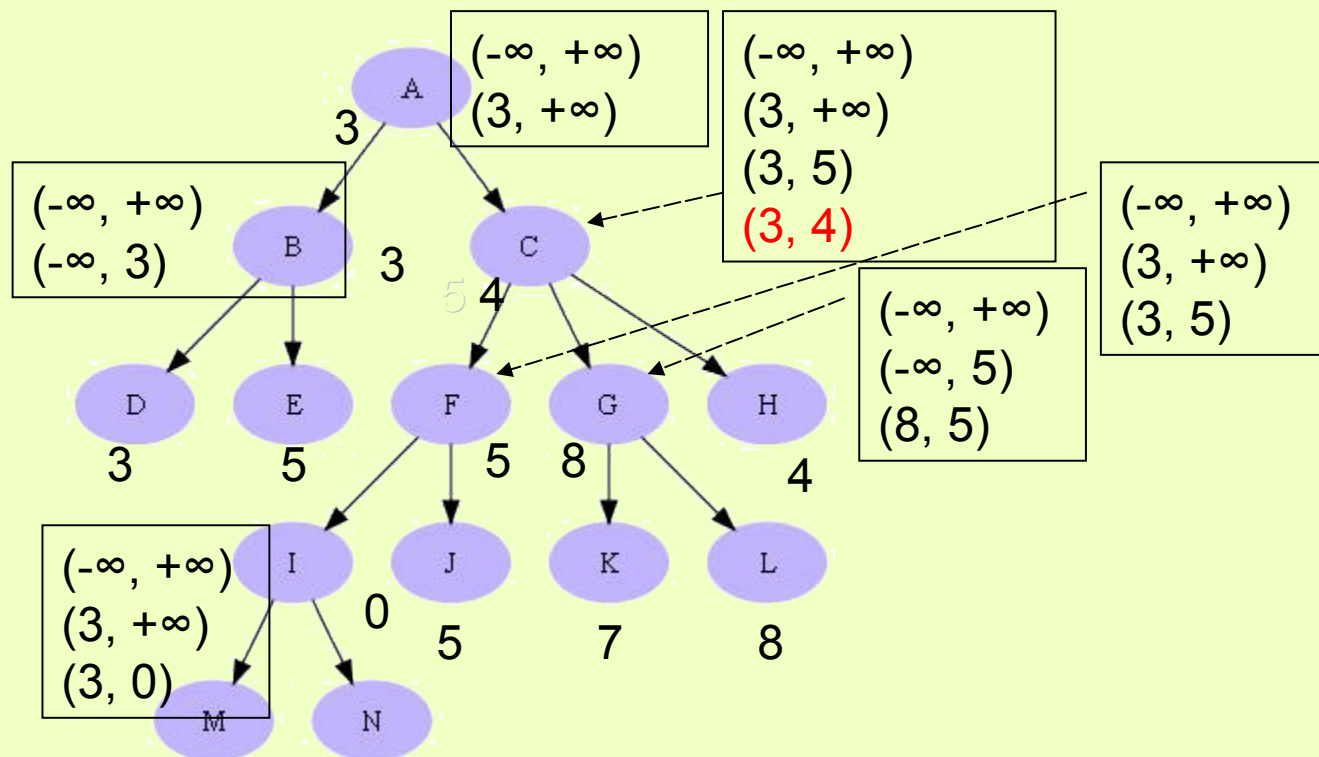
MAX( $\alpha$ )

MIN( $\beta$ )

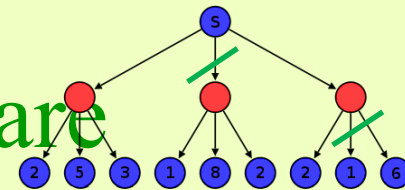
MAX( $\alpha$ )

MIN( $\beta$ )

MAX( $\alpha$ )



# Jocurile și căutarea – spațiul de căutare



Game's strategies → MiniMax with  $\alpha$ - $\beta$  pruning

□ Example

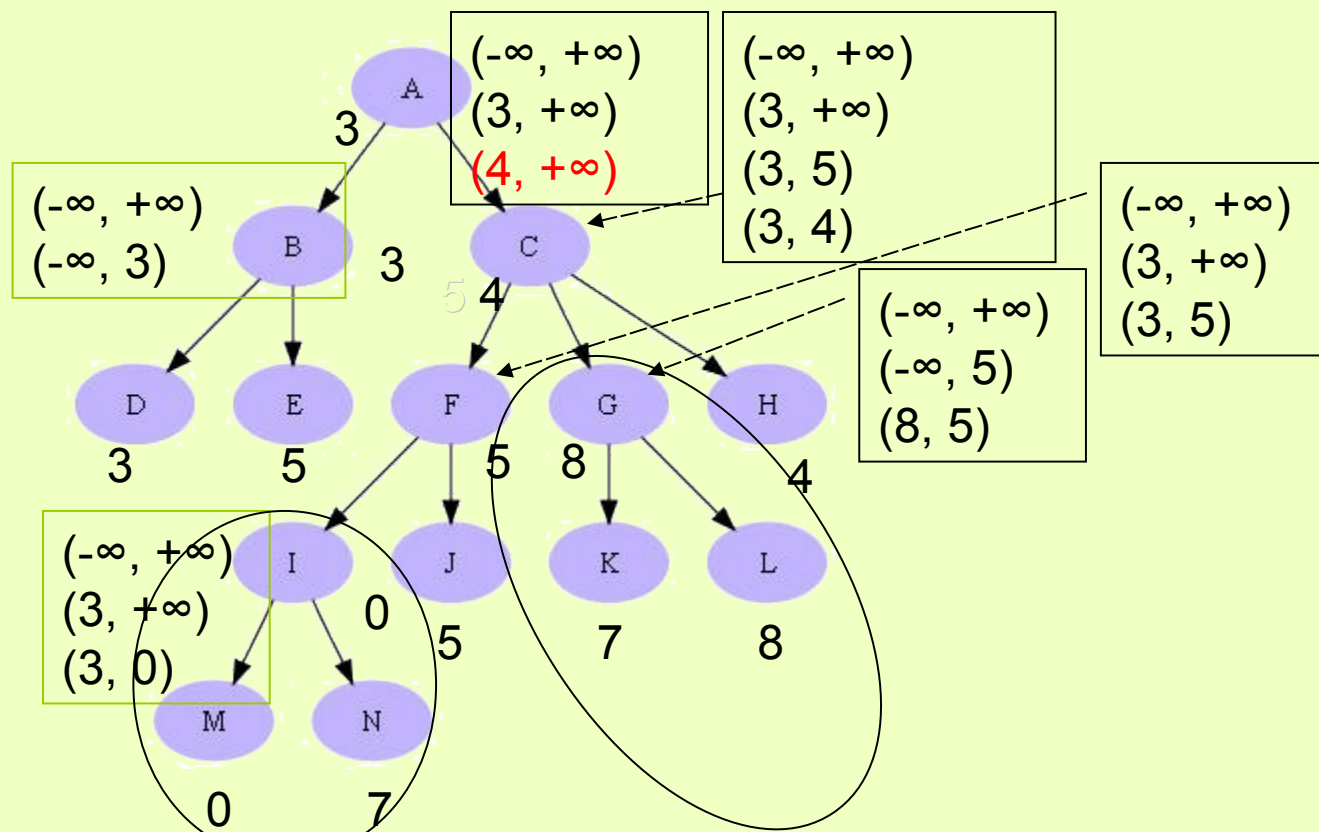
MAX( $\alpha$ )

MIN( $\beta$ )

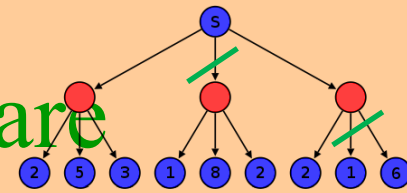
MAX( $\alpha$ )

MIN( $\beta$ )

MAX( $\alpha$ )



# Jocurile și căutarea – spațiul de căutare



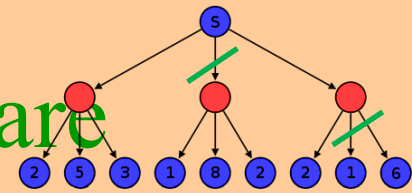
Game's strategies → MiniMax with  $\alpha$ - $\beta$  pruning

## □ Properties

- Pruning does not influence the final result
- A good sorting of the moves improves the pruning algorithm
- If the successors are perfectly sorted (the best are the first), then the time complexity is  $O(b^{d/2})$ , instead of  $O(b^d)$  (classic MiniMax)
- A beginner-level program can be transformed in an expert-level program



# Jocurile și căutarea – spațiul de căutare

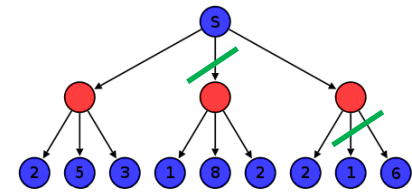


Game's strategies → MiniMax with  $\alpha$ - $\beta$  pruning

□ MinMax vs. MiniMax with  $\alpha$ - $\beta$  pruning

	MinMax	MinMax $\alpha$ - $\beta$
Time complexity	$O(b^m)$	$O(b^{m/2})$ with sorted nodes
Space complexity	$O(b^m)$	$O(2b^{d/2})$ – best case (when for a MAX node the first generated child is that of largest value / a MIN node the first generated child is that of smallest value) $O(b^d)$ – worst case (without pruning)
Completeness	Da	Yes
Optimality	Da	Yes

# Games and search – search space



- AndOr trees, MiniMax, MiniMax with  $\alpha$ - $\beta$  pruning
  - Large complexity → efficient search methods are required (for solving games)
    - Chess
      - $b \sim 35$
      - $d \sim 100$
      - $b^d \sim 35^{100} \sim 10^{154}$  nodes
    - Tic-Tac-Toe
      - $\sim 5$  legal moves (from 9)
      - $5^9 = 1\,953\,125$
      - $9! = 362\,880$  (if the computer moves first)
      - $8! = 40\,320$  (if the computer is the second player)
    - Go game
      - $b > 361$  (for a 19x19 board)

# Games and search – search space



## □ Reality

### ■ Checkers → Chinok

- Chinok beats Marion Tinsley in 1994 (after 40 years of challenges)
- A data base with final states computed for all positions of a perfect game of 8 pieces is utilized (444 billions of possible positions)

### ■ Chess → Deep Blue

- Deep Blue defeated Garry Kasparov in 1997
- 200 millions of positions are checked in a second
- Ramification factor is reduced to 6 by using  $\alpha$ - $\beta$  pruning (classic, 35-40)

### ■ Othello game

- Human players refuse to play against computers (because they are too good) – Moor (1980), Logistello (1997)

### ■ Go game

- Human players refuse to play against computers (because they are too weak)
- Ramification factor  $> 300 \rightarrow$  pattern-based algorithms are required

### ■ backgammon

- BKG (fuzzy logic), TD-Gammon (artificial neural networks)
- Computer defeated the world champion (because it was lucky)

# Games and search – search space



## □ Game's strategy

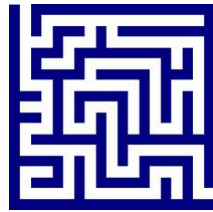
### ■ Step by step

- Ex.: XO, Checkers, Chess
- Algorithms – can work by:
  - **Linear structures**
    - Strategy of symmetry
    - Strategy of pairs
    - Parity strategy
    - Dynamic programming
    - Other strategies
  - Tree-based structures
    - AndOr trees
    - MiniMax (with Alpha-Beta cuttings)

### ■ Complete

- Ex.: labyrinth, map navigation
- Algorithm can identify
  - **An optimal path between 2 locations**
  - A sequence of actions for moving the player from a location into another location

# Games and search – search space



## Game's strategy → Path-finding

### □ Theoretical aspects

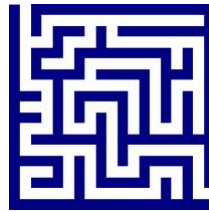
#### ■ Problem

- Identify on a map (possible, with obstacles) an optimal path between 2 given locations.
- Where?
  - On a map
    - Map as a matrix
    - Map as a graph (simple, mesh. quad-tree)
  - In an environment (known/unknown, static/dynamic)
- By who?
  - An agent
  - 2 agents
  - More agents

#### ■ Solution

- Using a search (optimization) method

# Games and search – search space



Game's strategy → Path-finding

## □ Theoretical aspects

### ■ Solution

#### □ How?

- Using a search (optimisation) method

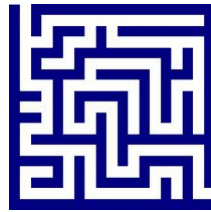
#### □ Which is the best solution?

- Computing speed
- Length of the path
- Quality of the path
- Available information

#### □ Difficulties

- Real-time solution
- Limited resources
- Very large search space
- Modelling the real-world involves uncertainty and heterogeneous elements (terrain, units, agents)

# Games and search – search space



Game's strategy → Path-finding

□ Search algorithms for maps represented as a matrix of tiles (tile mapped game) → labyrinths

■ Why?

□ Simplicity

■ Characteristics

□ Discrete search space

□ Tiles are square

□ Tiles can be traversed or blocked

□ Linear moves from a tile to another one (horizontal, vertical or diagonal moves)

■ Methods

□ A\*

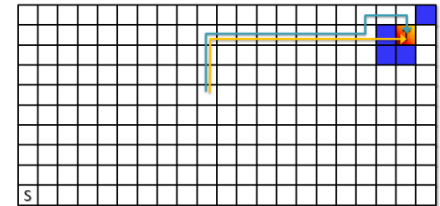
■ *Best-first search* algorithms

■ Represents a standard in game's industry

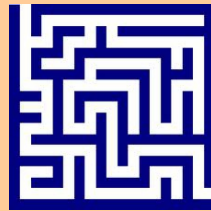
□ Heuristics → Manhattan (gen. of Minkowski norm)

■ Estimation of the distance between current point and destination

■ Ignore obstacles



# Games and search – search space



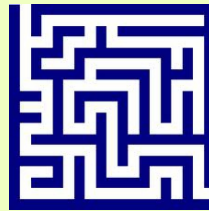
Game's strategy → Path-finding

- Search algorithms for maps represented as a graph
  - Why?
    - Matrix representation → large regions of the map have the same cost
  - Characteristics
    - Discrete search space
    - Tiles can have any shape
    - Tiles can be traversed or blocked
    - Random moves from a tile to another one
  - Methods
    - Paths of edges
    - Paths of nodes
      - Points
      - Edges of a polygon
      - Middle of the polygon
    - Eg. Dijkstra, Floyd-Warshall, Kruskal, etc



# Games and search – search space

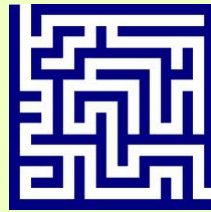
---



Game's strategy → Path-finding

- Improvements of the search algorithms
  - Better heuristics
  - Hierarchical abstracting
  - Decentralised approaches

# Games and search – search space



Game's strategy → Path-finding

## □ Improvements of the search algorithms

### ■ Better heuristics

#### □ Heuristics without memory

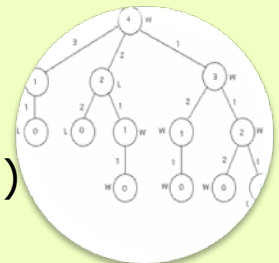
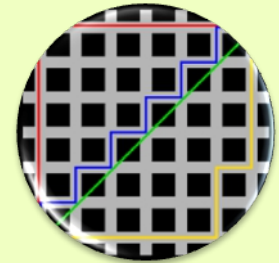
- Eg. Manhattan
- Very fast to compute
- Do not require supplementary memory
- Good quality (sometimes)

#### □ Heuristics based on memory

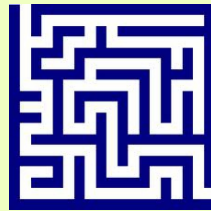
- Eg. Heuristics based on landmarks
- Fast enough to compute
- Require memory
- Good quality (identification of dead ends)

#### □ Imperfect information

- Very expansive to compute and store
- Very fast to utilize (after they have been computed)
- Impractical



# Jocurile și căutarea – spațiul de căutare



Game's strategy → Path-finding

## □ Improvements of the search algorithms

### ■ Hierarchical abstracting

#### □ Graphs where

- Flows of motion are annotated
- Domination relations are associated to edges
- Levels are identified (eg. Room, house, town, city)

### ■ Decentralised approaches

#### □ Main idea:

- Decompose the problem in sub-problems that
  - Can be completely solved
  - Can be sub-optimal
  - Can be incomplete

#### □ Aim

- All the elements (collaborative identification) have to arrive at destination

#### □ Difficulties

- Increased search space:  $O(n) \rightarrow O(n^m)$
- Ramification factor explodes (from 4 or 8 to  $5^m$  or  $9^m$ )

# Jocurile și căutarea – spațiul de căutare



## Game's strategy → Planning

- Theoretical aspects
  - Problem
    - Planning (sorting) some actions
      - Moves, selections, rotations, etc.
    - Input:
      - An initial state



# Jocurile și căutarea – spațiul de căutare



## Game's strategy → Planning

- Theoretical aspects
  - Problem
    - Planning (sorting) some actions
      - Moves, selections, rotations, etc.
    - Input:
      - An initial state
      - A final state

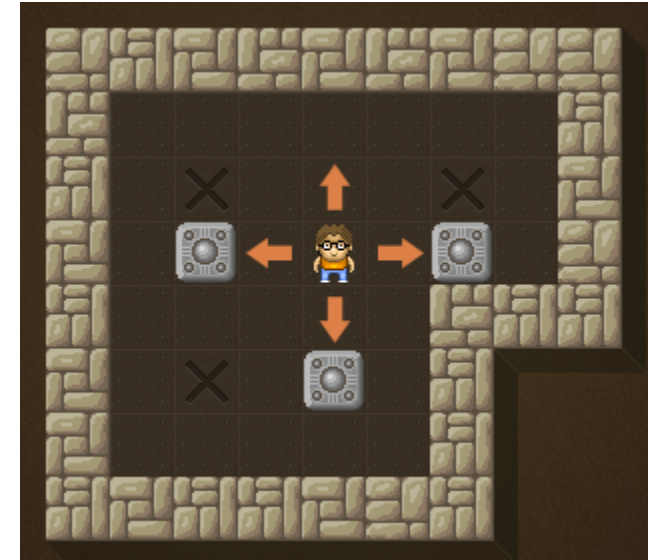


# Games and search – search space



## Game's strategy → Planning

- Theoretical aspects
  - Problem
    - Planning (sorting) some actions
      - Moves, selections, rotations, etc.
    - Input:
      - An initial state
      - A final state
      - A set of possible actions

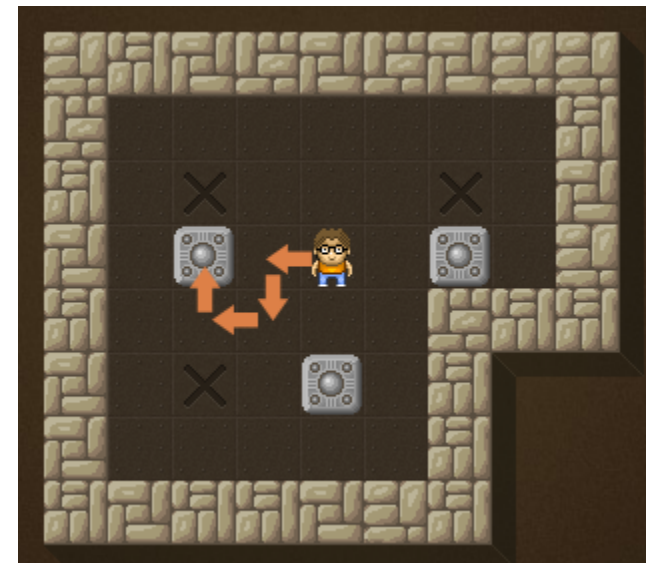


# Games and search – search space



## Game's strategy → Planning

- Theoretical aspects
  - Problem
    - Planning (sorting) some actions
      - Moves, selections, rotations, etc.
    - Input:
      - An initial state
      - A final state
      - A set of possible actions
    - Output:
      - Identify a sequence of actions that transforms the initial state into the final state



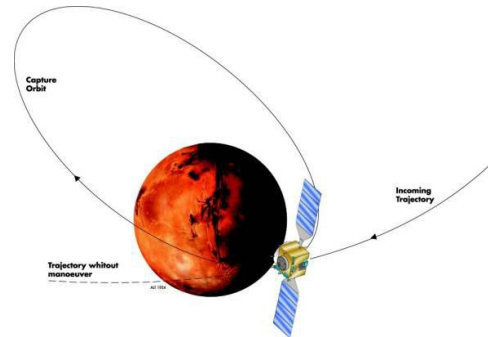
# Games and search – search space



## Game's strategy → Planning

### □ Why?

- NPC games (*First-person shooter, Role-playing, Real-time strategy*)
- Real-world planning problems
  - Space telescope
  - Robots
  - UAVs





# Games and search – search space



## Game's strategy → Planning

### □ How?

#### ■ Algorithms

- For selecting an action (shooting)
- For evaluating the environment

#### ■ Approaches

- STRIPS →

<http://www.dis.uniroma1.it/~degiacon/didattica/dottorato-stavros-vassos/>

- HTN

#### ■ Behaviour simulation

- Finite state machines
- Behaviour trees
- GOAP (FEAR)

# Review



## □ Game's definition

- Initial state (how the elements are initially located)
- Possible actions (allowed moves)
- Final test (when a game ends)
- Utility functions (who wins and what is the profit)

## □ Solving strategy for games

- Based on (almost) complete exploration of the search tree
  - AndOr → who wins
  - MiniMax → who wins and what is its profit
  - MiniMax with  $\alpha$ - $\beta$  pruning → who wins and what is its profit and what moves do not deserve to be performed
- Based on intelligent strategies
  - Symmetry
  - Pairs
  - Parity
  - Dynamic programming
  - A\* (Path-finding, Planning)

# Next lecture

---

- A. Short introduction in Artificial Intelligence (AI)
- B. Solving search problems
  - A. Definition of search problems
  - B. Search strategies
    - A. Uninformed search strategies
    - B. Informed search strategies
    - C. Local search strategies (Hill Climbing, Simulated Annealing, Tabu Search, Evolutionary algorithms, PSO, ACO)
    - D. Adversarial search strategies
- C. Intelligent systems**
  - A. Rule-based systems in certain environments**
  - B. Rule-based systems in uncertain environments (Bayes, Fuzzy)
  - C. Learning systems
    - A. Decision Trees
    - B. Artificial Neural Networks
    - C. Support Vector Machines
    - D. Evolutionary algorithms
  - D. Hybrid systems

# Next lecture - useful information

---

- ❑ Chapter III of *S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall, 1995*
- ❑ Chapter 4 and 5 of *H.F. Pop, G. Șerban, Inteligență artificială, Cluj Napoca, 2004*
- ❑ Chapter 2 of *Adrian A. Hopgood, Intelligent Systems for Engineers and Scientists, CRC Press, 2001*
- ❑ Chapter 6 and 7 of *C. Groșan, A. Abraham, Intelligent Systems: A Modern Approach, Springer, 2011*

- 
- Presented information have been inspired from different bibliographic sources, but also from past AI lectures taught by:
    - PhD. Assoc. Prof. Mihai Oltean – [www.cs.ubbcluj.ro/~moltean](http://www.cs.ubbcluj.ro/~moltean)
    - PhD. Assoc. Prof. Crina Groșan - [www.cs.ubbcluj.ro/~cgrosan](http://www.cs.ubbcluj.ro/~cgrosan)
    - PhD. Prof. Horia F. Pop - [www.cs.ubbcluj.ro/~hfpop](http://www.cs.ubbcluj.ro/~hfpop)