



UNIVERSITATEA BABEȘ-BOLYAI  
Facultatea de Matematică și Informatică



# INTELIGENȚĂ ARTIFICIALĂ

---

**Rezolvarea problemelor de căutare**

Strategii de căutare adversială

Laura Dioșan

# Sumar

---

## A. Scurtă introducere în Inteligența Artificială (IA)

## B. Rezolvarea problemelor prin căutare

- Definirea problemelor de căutare
- Strategii de căutare
  - Strategii de căutare neinformate
  - Strategii de căutare informate
  - Strategii de căutare locale (Hill Climbing, Simulated Annealing, Tabu Search, Algoritmi evolutivi, PSO, ACO)
  - Strategii de căutare adversială

## C. Sisteme inteligente

- Sisteme care învață singure
  - Arbori de decizie
  - Rețele neuronale artificiale
  - Mașini cu suport vectorial
  - Algoritmi evolutivi
- Sisteme bazate pe reguli
- Sisteme hibride

# Materiale de citit și legături utile

---

- ❑ capitolul II.5 din *S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall, 1995*
- ❑ capitolul 6 din *H.F. Pop, G. Șerban, Inteligență artificială, Cluj Napoca, 2004*
- ❑ documentele din directorul *06\_adversial\_minimax*

# Conținut

## Jocuri

- Un pic de istorie
- Câteva repere teoretice
- Jocurile și căutarea
  - Definiții, componente, clasificare
  - Strategii și algoritmi de căutare



# Jocuri – un pic de istorie

---

- ❑ Provocări ale inteligenței
  - Dame (Mesopotamia, cc. 3000 î.Hr.)
  - Go (China, sec. VI î.Hr.)
  - Sah (India, sec. VI d.Hr.)
- ❑ Tradițional, jocurile erau văzute formal, ca o extensie a algoritmilor de căutare
  - Căutarea clasică: un singur agent, încearcă fără piedici să își atingă obiectivul
  - Jocurile: căutare în prezența unui adversar (agent ostil) + aduce incertitudine
- ❑ Jocurile și IA
  - Proiectarea și testarea algoritmilor
  - Unul dintre cele mai vechi domenii ale IA

# Jocuri – un pic de istorie

---

## □ Jucarea jocurilor

### ■ De către oameni

- Activitate inteligentă
- Compararea aptitudinilor

### ■ De către calculatoare

- Mediu propice pentru dezvoltarea tehnicilor de IA
  - Joc = problemă
    - structurată (succes sau eșec)
    - rezolvabilă prin căutare (simplă sau euristică)
- Limite

# Jocuri – câteva repere teoretice

---

- ❑ Probleme dificile cu o structură inițială minimă de cunoștințe
- ❑ Stări și acțiuni ușor de reprezentat
- ❑ Puține cunoștințe necesare despre mediu
- ❑ Jocurile sunt un caz particular al problemelor de căutare
- ❑ Deseori au spații de căutare foarte mari
  - Complexitatea jocurilor → incertitudine
    - ❑ datorată insuficienței de timp pentru a calcula consecințele tuturor mutărilor și
    - ❑ nu lipsei de informație
- ❑ Jocurile sunt interesante 😊

# Jocurile și căutarea

---

- Formalizare
- Reprezentarea jocurilor
- Tipuri de jocuri
- Spațiul de căutare
  - Reprezentare
  - Metode de explorare



# Jocurile și căutarea – formalizare

---

- Rezolvarea unei probleme de căutare
  - Etapa x: definirea spațiului de căutare
    - Spații liniare
    - Spații arborescente
      - Arbori
      - Grafe
  - Etapa x + 1: alegerea unei strategii de căutare
    - Care mutare/strategie?
      - Care determină câștigarea jocului
      - Care poate fi determinată cât mai repede (complexitate temporală redusă)
      - Care poate fi determinată cu efort fizic cât mai mic (complexitate spațială redusă)
- Problema:
  - cum se poate determina cea mai bună mutare următoare într-un timp cât mai scurt?
- O soluție:
  - căutarea între mutările posibile și consecințele lor

# Jocurile și căutarea – formalizare

- Căutarea adversială este folosită în jocurile în care unii jucători încearcă să-și maximizeze scorul, dar se confruntă cu unul sau mai mulți adversari
- Reprezentarea jocurilor ca probleme de căutare
  - **Stări**
    - configurațiile (tablei) de joc + jucătorul care trebuie să mute
    - totalitatea stărilor posibile → arborele de căutare
  - **Operatori (acțiuni, funcție succesori)**
    - mutările permise
  - **Starea inițială**
    - configurația inițială a jocului
  - **Starea scop (finală)**
    - configurația (câștigătoare) care termină jocul
  - **Funcție de utilitate (funcție scor)**
    - asociază o valoare numerică unei stări
- Dificultăți
  - Jocurile pot fi probleme de căutare foarte dificile
    - totuși ușor de formalizat
  - Găsirea soluției optime poate fi nefezabilă
    - o soluție care învinge adversarul este acceptabilă
    - o soluție „inacceptabilă” nu numai că induce costuri mai mari, dar provoacă înfrângerea

# Jocurile și căutarea – formalizare

---

## □ Definiții cheie

### ■ Situație conflictuală

- Situație în care acționează mai multe părți care au scopuri contrare
- Consecința acțiunii unei părți depinde de reacția celorlalte părți

### ■ Joc

- Modelare simplificată a unei situații conflictuale
- Însiruire de decizii (acțiuni, mutări) luate de părți cu interese contrastante

### ■ Mutare

- Funcție  $H : \{\text{pozițiile jocului}\} \rightarrow \{\text{pozițiile jocului}\}$
- Dacă  $p$  – o poziție în joc,  $H(p)$  – o nouă poziție

### ■ Reguli

- Sistem de condiții privind mutările posibile

### ■ Strategie

- Ansamblu de reguli care definesc mutările libere în funcție de situația concretă ivită

# Jocurile și căutarea – reprezentare

---

## □ Componente

- Jucători
- Mutări (strategii)
- Criterii de câștig

## □ Forme de reprezentare pentru un joc

- Forma strategică → matrice
  - Jucătorii
  - Strategiile
  - Câștigurile asociate fiecărui jucător și fiecărei strategii
- Forma extinsă → arbore
  - Nivel → jucător
  - Nod → alegere (mutare)

# Jocurile și căutarea – reprezentare



## □ Forma strategică → matrice

- Ex. Dilema prizonierului: doi prizonieri sunt chestionați de poliție. Poliția știe ceva despre ceea ce au făcut ei, dar nu are toate informațiile. Pentru a afla, cei doi prizonieri sunt închiși în două celule și sunt interogați. Prizonierii au 2 opțiuni:
  - Pot spune toată povestea (pot să se trădeze unul pe altul)
  - Pot să nu spună nimic (pot să coopereze)

Nici un prizonier nu știe ce va răspunde celălalt. În funcție de răspunsurile lor, pedepsele sunt următoarele:

- Dacă amândoi tac (cooperează), sentința este ușoară (1 an fiecare)
- Dacă unul vorbește (trădează) și unul tace (cooperează), trădătorul este eliberat, iar pârâtul primește 10 ani
- Dacă amândoi vorbesc (se trădează reciproc), sentința este de 5 ani fiecare

Ce vor face cei doi prizonieri?

J1 \ J2	Cooperare	Trădare
Cooperare	(1,1)	(10,0)
Trădare	(0,10)	(5,5)

# Jocurile și căutarea – reprezentare



## □ Forma strategică → matrice

- Ex. Vânătoarea de cerbi: doi indivizi merg la vânătoare. Fiecare poate alege să vâneze:
  - un cerb sau
  - un iepure,
- fără însă să știe ce a ales celălalt individ. Vânărea
  - cerbilor, mai profitoare (câștig=4), se poate face doar în doi,
  - iepurilor, mai puțin valoroasă (câștig=1), se poate face individual.
- Ce vor alege să vâneze cei doi indivizi?

J1 \ J2	Cerb	Iepure
Cerb	(4,4)	(1,3)
Iepure	(3,1)	(3,3)

# Jocurile și căutarea – reprezentare

## □ Forma strategică → matrice

### ■ Ex. economic

- Reclama făcută de 2 firme pe aceeași piață
- Înțelegerile la nivel de cartel pentru stabilirea prețurilor

### ■ Ex. sportiv



- Doi cicliști aflați în fața plutonului poartă consecutiv trena (cooperează) pentru a nu fi ajunși din urmă. De multe ori, doar unul duce trena (cooperează), iar la linia de sosire este trădat de adversar.

### ■ Ex. sociologic



- Când cunoaștem o nouă persoană, tindem să fim foarte atenți pentru a avea o poziție de siguranță (competiție). Ambele persoane pot semnală dorința de a se muta de la pozițiile defensive către
  - interacțiune și
  - recunoașterea unei intenții comune.

# Jocurile și căutarea - tipologie

---

- Numărul jucătorilor
  - 1
  - 2
  - Mai mulți
  
- Comunicarea între jucători
  - Cooperative
  - Ne-cooperative
  
- Strategiile de joc urmate
  - Simetrice
  - Asimetrice



# Jocurile și căutarea – tipologie

---

## □ Câștigurile jucătorilor

### ■ Sumă zero

- câștigul unui jucător = pierderea celorlați jucători → un jucător trebuie să câștige sau jocul se sfârșește cu remiză

### ■ Sumă diferită de zero

- câștigul unui jucător  $\neq$  pierderea celorlați jucători

## □ Natura mutărilor efectuate

### ■ Cu mutări libere

- mutarea este aleasă conștient dintr-o mulțime de acțiuni posibile → jocuri deterministe

### ■ Cu mutări întâmplătoare

- factor aleator (zar, cărți de joc, monede) → jocuri non-deterministe

# Jocurile și căutarea – tipologie

---

## □ Informațiile despre joc

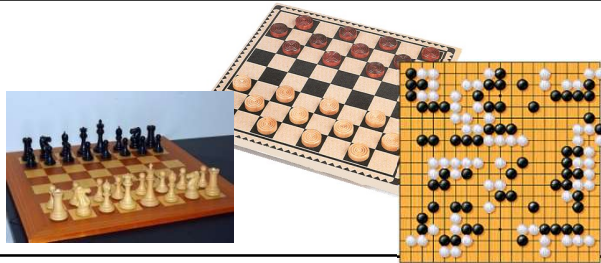
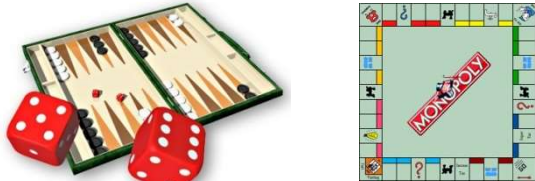
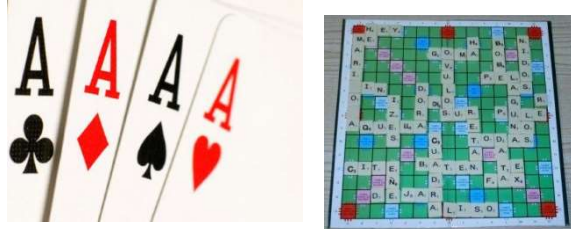
### ■ Cu informație perfectă

- un jucător, înainte de a executa o mutare, cunoaște rezultatele tuturor mutărilor precedente (ale lui și ale adversarilor);
- de obicei, jocul e cu mutări secvențiale

### ■ Cu informație imperfectă

- un jucător nu cunoaște toate efectele mutărilor precedente

# Jocurile și căutarea – tipologie

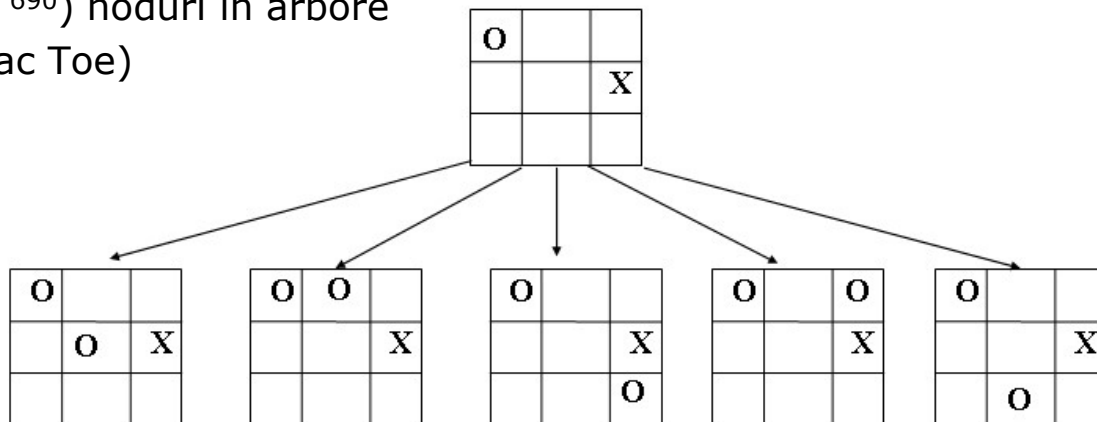
	Deterministic	Non-deterministic (aleator)
Informație perfectă		
Informație imperfectă	Vaporașe/Avioane/ Submarine	

# Jocurile și căutarea – spațiul de căutare



## □ Reprezentare

- Spațiu liniar
- Spațiu arborescent → Arborele jocului
  - Identificarea strategiei de câștig → explorarea întregului arbore
  - foarte mare pt anumite jocuri
    - Șah („drosfila IA”)
      - factor de ramificare  $\approx 35$
      - $\approx 50$  de mutări pe jucător
      - $\approx 35^{100}$  ( $10^{154}$ ) noduri
      - $10^{40}$  noduri distincte (dimensiunea grafului de căutare)
    - Go
      - $\approx 200$  mutări/stare, 300 niveluri
      - $200^{300}$  ( $10^{690}$ ) noduri în arbore
  - exemplu – XO (Tic Tac Toe)



# Jocurile și căutarea – spațiul de căutare



## □ Strategia de joc (a unui jucător)

### ■ Definire

- ansamblul mutărilor unui jucător care
- ține cont de:
  - regulile jocului
  - starea curentă a jocului
- ≠ mutare

### ■ Tipologie

- Scop
  - Strategii pas cu pas
    - În fiecare etapă a jocului se identifică mutarea cea mai bună
  - Strategii complete
    - Se identifică o succesiune de mutări

# Jocurile și căutarea – spațiul de căutare



## □ Strategia de joc

### ■ Pas cu pas

- Ex.: XO, Dame, Șah
- Algoritmi – pot lucra cu structuri:
  - Liniare
    - Strategia simetriei
    - Strategia perechilor
    - Strategia parității
    - Programare dinamică
    - Alte strategii
  - Arborescente
    - Arbori AndOr
    - MiniMax (cu tăieturi Alpha-Beta)

### ■ Completă

- Ex.: ieșirea dintr-un labirint, deplasarea pe o hartă între 2 locații date
- Algoritmi pot să identifice
  - Un drum optim de la o locație la alta
  - O succesiune de acțiuni care să deplaseze jucătorul de la o locație la alta

# Jocurile și căutarea – spațiul de căutare



## □ Strategia de joc

### ■ Pas cu pas

- Ex.: XO, Dame, Șah
- Algoritmi – pot lucra cu structuri:

#### ■ **Liniare**

- Strategia simetriei
- Strategia perechilor
- Strategia parității
- Programare dinamică
- Alte strategii

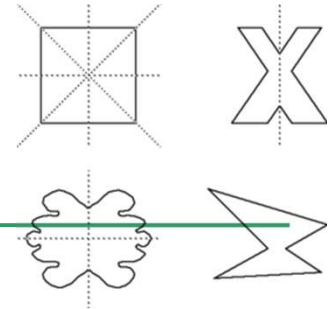
#### ■ Arborescente

- Arbori AndOr
- MiniMax (cu tăieturi Alpha-Beta)

### ■ Completă

- Ex.: ieșirea dintr-un labirint, deplasarea pe o hartă între 2 locații date
- Algoritmi pot să identifice
  - Un drum optim de la o locație la alta
  - O succesiune de acțiuni care să deplaseze jucătorul de la o locație la alta

# Jocurile și căutarea – spațiul de căutare



## Strategii de joc → Strategia ...

### □ Exemplu: jocul *hașurează căsuțe*

#### ■ Se dă:

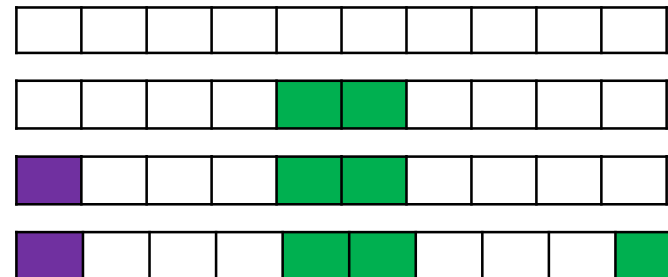
- O bandă de hârtie este împărțită în  $n$  căsuțe. Alternativ doi jucători A și B hașurează câte maxim  $k$  căsuțe adiacente, nehașurate ( $n$  și  $k$  au aceeași paritate). Cel care nu mai poate muta pierde. Inițial mută jucătorul A.

#### ■ Se cere:

- Să se determine dacă jucătorul A are strategie sigură de câștig. În caz afirmativ, să se programeze mutările lui A, cele ale lui B fiind citite de la tastatură.

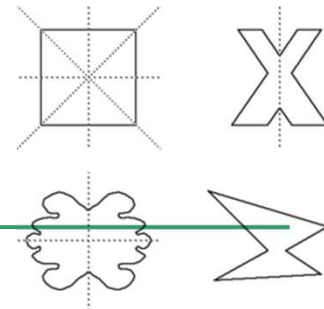
#### ■ Caz concret:

- $n = 10, k = 4$





# Jocurile și căutarea – spațiul de căutare



## Strategii de joc → Strategia simetriei

### ■ Exemplu: jocul *hașurează căsuțe*

#### ■ Se dă:

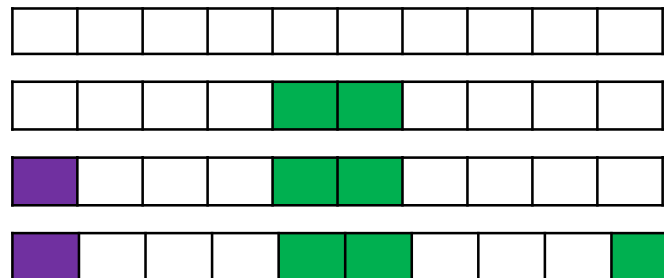
- O bandă de hârtie este împărțită în  $n$  căsuțe. Alternativ doi jucători A și B hașurează câte maxim  $k$  căsuțe adiacente, nehașurate ( $n$  și  $k$  au aceeași paritate). Cel care nu mai poate muta pierde. Inițial mută jucătorul A.

#### ■ Se cere:

- Să se determine dacă jucătorul A are strategie sigură de câștig. În caz afirmativ, să se programeze mutările lui A, cele ale lui B fiind citite de la tastatură.

#### ■ Caz concret:

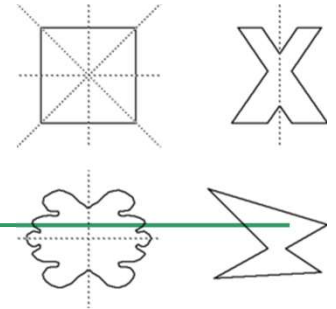
- $n = 10, k = 4$



#### ■ Soluție

- Dacă  $n$  – nr. par
  - Prima mutare → jucătorul A hașurează un nr. par de căsuțe din mijlocul benzii
  - La următoarele mutări jucătorul A îl imită pe jucătorul B simetric față de mijlocul benzii
- Dacă  $n$  – nr. impar
  - Prima mutare → jucătorul A hașurează un nr. impar de căsuțe din mijlocul benzii
  - La următoarele mutări jucătorul A îl imită pe jucătorul B simetric față de mijlocul benzii

# Jocurile și căutarea – spațiul de căutare



## Strategii de joc → Strategia simetriei

### □ Aspecte teoretice

- Jucătorul B imită mutările jucătorului A pe baza unei (unui) axe (centru) de simetrie
- Dacă A mai poate muta, atunci și B mai poate muta
  - Jocul se termină când A nu mai poate muta
- E posibil ca prima mutare să nu aibă simetric

# Jocurile și căutarea – spațiul de căutare



## Strategii de joc → Strategia ...

### ■ Exemplu - Jocul *pătratelor alunecătoare*

#### ■ Se dă:

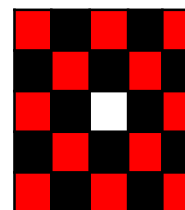
- În fiecare căsuță a unei table pătratice  $n \times n$  ( $n$  – nr. impar) se află un pătrat roșu sau negru astfel încât tabla se aseamănă cu o tablă de șah. Căsuța din mijlocul tablei este goală (nu conține un pătrat). Alternativ, doi jucători A (cu pătratele roșii) și B (cu pătratele negre) mută câte unul din pătratele lor în căsuța liberă de pe tablă. Pătratul mutat trebuie să se afle inițial într-o căsuță vecină (pe orizontală sau verticală) cu căsuța liberă. Jucătorul care nu mai poate muta nici un pătrat de-al lui pierde jocul. Jucătorul B mută primul.

#### ■ Se cere:

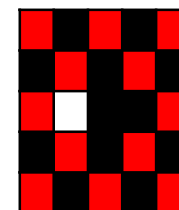
- Să se determine dacă jucătorul A are strategie sigură de câștig. În caz afirmativ, să se programeze mutările lui A, cele ale lui B fiind citite de la tastatură.

#### ■ Caz concret:

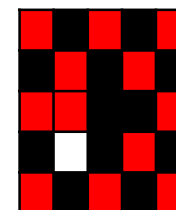
- $n = 5$  (jocul propus inițial de către G.W.Lewthwaite)



a



b



c

# Jocurile și căutarea – spațiul de căutare



## Strategii de joc → Strategia perechilor

### □ Exemplu - Jocul *pătratelor alunecătoare*

#### ■ Se dă:

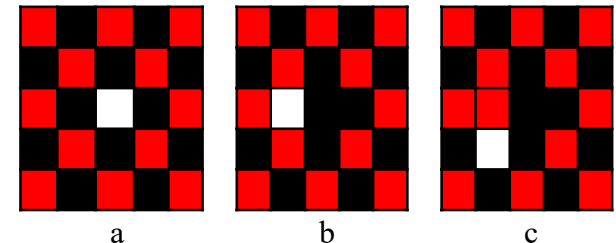
- În fiecare căsuță a unei table pătratice  $n*n$  ( $n$  – nr. impar) se află un pătrat roșu sau negru astfel încât tabla se aseamănă cu o tablă de șah. Căsuța din mijlocul tablei este goală (nu conține un pătrat). Alternativ, doi jucători A (cu pătratele roșii) și B (cu pătratele negre) mută câte unul din pătratele lor în căsuța liberă de pe tablă. Pătratul mutat trebuie să se afle inițial într-o căsuță vecină (pe orizontală sau verticală) cu căsuța liberă. Jucătorul care nu mai poate muta nici un pătrat de-al lui pierde jocul. Jucătorul B mută primul.

#### ■ Se cere:

- Să se determine dacă jucătorul A are strategie sigură de câștig. În caz afirmativ, să se programeze mutările lui A, cele ale lui B fiind citite de la tastatură.

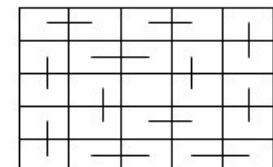
#### ■ Caz concret:

- $n = 5$  (jocul propus inițial de către G.W.Lewthwaite)



#### ■ Soluție

- Se împarte tabla în *Domino-uri*, căsuța din mijlocul tablei (inițial goală) nefacând parte din nici un *Domino*.
- După fiecare mutare a jucătorului B, căsuța liberă se află acoperită de același *Domino* care acoperă și o piesă de culoare roșie. Această piesă o va muta A, având în acest fel întotdeauna ceva de mutat.
- acoperirea cu *Domino-uri* în spirală, plecându-se din colțul stânga sus, spre dreapta



# Jocurile și căutarea – spațiul de căutare

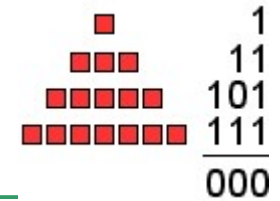


## Strategii de joc → Strategia perechilor

### □ Aspecte teoretice

- Generalizare a strategiei simetriei
- Gruparea mutărilor în perechi:
  - (mutare jucător A, mutare jucător B)
- Dacă A mai poate muta, atunci și B mai poate muta
  - Jocul se termină când A nu mai poate muta
- E posibil ca prima mutare să nu poată fi împerecheată

# Jocurile și căutarea – spațiul de căutare



## Strategii de joc → Strategia parității

### ■ Exemplu - Jocul *NIM*

#### ■ Se dau:

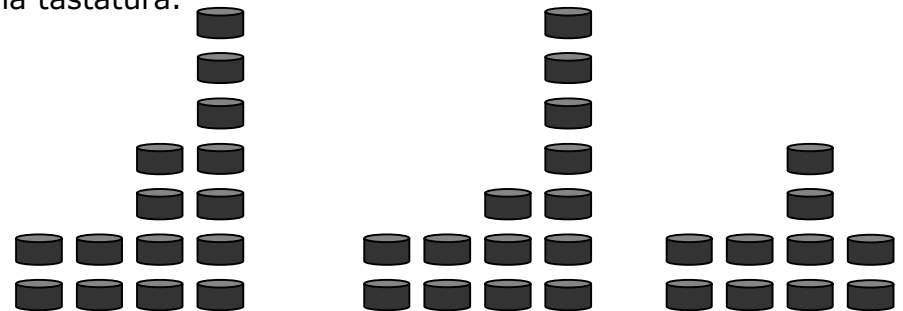
- $N$  stive, fiecare conținând  $p_i$  obiecte. 2 jucători, A și B, extrag, alternativ, dintr-o singură stivă, oricâte obiecte. Jucătorul care execută ultima extragere câștigă jocul. Jucătorul A mută primul.

#### ■ Se cere:

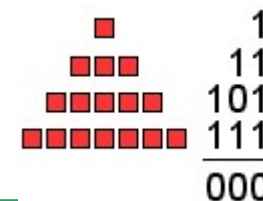
- Să se determine dacă jucătorul A are strategie sigură de câștig. În caz afirmativ, să se programeze mutările lui A, cele ale lui B fiind citite de la tastatură.

#### ■ Exemplu:

- 4 stive cu 2, 2, 4 și, respectiv, 7 obiecte



# Jocurile și căutarea – spațiul de căutare



## Strategii de joc → Strategia parității

### ■ Exemplu - Jocul *NIM*

#### ■ Se dau:

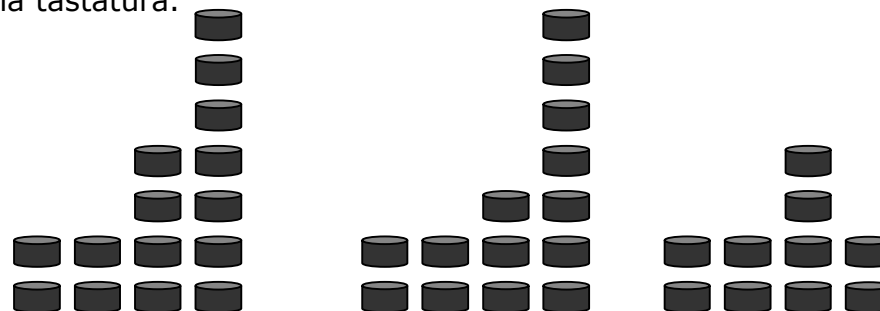
- $N$  stive, fiecare conținând  $p_i$  obiecte. 2 jucători, A și B, extrag, alternativ, dintr-o singură stivă, oricâte obiecte. Jucătorul care execută ultima extragere câștigă jocul. Jucătorul A mută primul.

#### ■ Se cere:

- Să se determine dacă jucătorul A are strategie sigură de câștig. În caz afirmativ, să se programeze mutările lui A, cele ale lui B fiind citite de la tastatură.

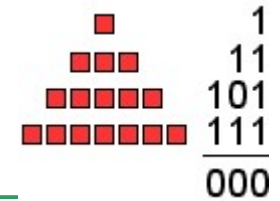
#### ■ Exemplu:

- 4 stive cu 2, 2, 4 și, respectiv, 7 obiecte



#### ■ Soluție

- Nr de obiecte din fiecare stivă se reprezintă binar (ca sumă a puterilor lui 2)
  - fiecare număr natural admite o descompunere unică ca sumă de puteri distincte ale lui 2
- Stare pară – toate puterile lui 2 din reprezentarea jocului apar de un număr par de ori (în toate stivele)
- Stare impară – orice stare care nu este pară
  - $T1 \rightarrow$  dintr-o stare impară se ajunge într-o stare pară printr-o mutare convenabilă
  - $T2 \rightarrow$  dintr-o stare pară se ajunge într-o stare impară prin orice fel de mutare
- Extragerea de pe stiva care conține cel mai semnificativ bit pe poziția  $k$ 
  - $K$  poziția celui mai semnificativ bit în suma NIM a tuturor stivelor



# Jocurile și căutarea – spațiul de căutare

## Strategii de joc → Strategia parității

### □ Aspecte teoretice

#### ■ 2 tipuri de poziții în joc

- Poziție pară (singulară)
- Poziție impară (nesingulară)

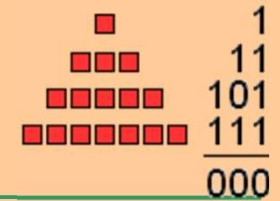
#### ■ Teoreme

- T1: poziția impară, printr-o mutare **convenabilă**, → poziție pară (jucătorul A)
- T2: poziția pară, prin **orice** mutare, → poziție impară (jucătorul B)

#### ■ Câștigător

- → poziția finală = poziție pară





# Jocurile și căutarea – spațiul de căutare

## Strategii de joc → Strategia parității

### □ Exemplu - Jocul *NIM*

#### ■ Demonstrarea celor 2 teoreme T1 și T2

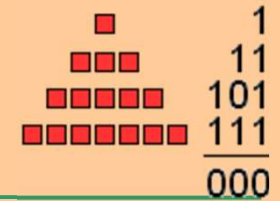
- $S_1: 2 = 2^1, S_2: 3 = 2^1 + 2^0, S_3: 4 = 2^2, S_4: 5 = 2^2 + 2^0$
- $2^0 - 2, 2^1 - 2, 2^2 - 2 \rightarrow$  stare pară

#### □ Extragerea dintr-o stivă cu număr par de obiecte

- A unui număr par de obiecte (ex. 2 obiecte din  $S_3$ ) = >
  - $S_1: 2 = 2^1, S_2: 3 = 2^1 + 2^0, S_3: 4-2 = 2^1, S_4: 5 = 2^2 + 2^0$
  - $2^0 - 2, 2^1 - 3, 2^2 - 1 \rightarrow$  stare impară
- A unui număr impar de obiecte (ex. 1 obiect din  $S_1$ )
  - $S_1: 2-1 = 2^0, S_2: 3 = 2^1 + 2^0, S_3: 4 = 2^2, S_4: 5 = 2^2 + 2^0$
  - $2^0 - 3, 2^1 - 1, 2^2 - 2 \rightarrow$  stare impară

#### □ Extragerea dintr-o stivă cu număr impar de obiecte

- A unui număr par de obiecte (ex. 2 obiecte din  $S_4$ ) = >
  - $S_1: 2 = 2^1, S_2: 3 = 2^1 + 2^0, S_3: 4 = 2^2, S_4: 5-2 = 2^1 + 2^0$
  - $2^0 - 2, 2^1 - 3, 2^2 - 1 \rightarrow$  stare impară
- A unui număr impar de obiecte (ex. 1 obiect din  $S_2$ )
  - $S_1: 2 = 2^1, S_2: 3-1 = 2^1, S_3: 4 = 2^2, S_4: 5 = 2^2 + 2^0$
  - $2^0 - 1, 2^1 - 2, 2^2 - 2 \rightarrow$  stare impară



# Jocurile și căutarea – spațiul de căutare

## Strategii de joc → Strategia parității

### □ Exemplu - Jocul *NIM*

#### ■ algoritm

#### □ Pp. următorul joc:

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

#### □ Pași:

#### 1. Se calculează suma Nim ( $S_{Nim}$ ) a tuturor stivelor

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

#### 2. Se identifică poziția $k$ a celui mai reprezentativ 1 în suma Nim

▪  $S_{Nim} = 2 = 010_{(2)} \rightarrow$  poziția a 2-a ( $k = 2$ )

#### 3. Se caută stiva cu cel mai reprezentativ bit pe poziția $k$ (stiva care descrește dacă se face XOR între ea și suma Nim)

▪  $S_1: 3 = 010$ ,  $S_2: 4 = 100$ ,  $S_3: 5 = 101 \Rightarrow S_1$  sau

▪  $S_1 \text{ XOR } S_{Nim} = 3 \text{ XOR } 2 = 011 \text{ XOR } 010 = 001 = 1$

▪  $S_2 \text{ XOR } S_{Nim} = 4 \text{ XOR } 2 = 100 \text{ XOR } 010 = 110 = 6$

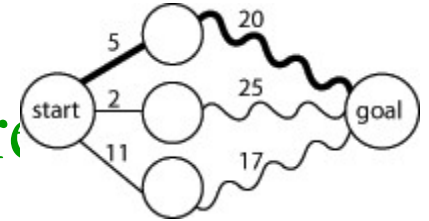
▪  $S_3 \text{ XOR } S_{Nim} = 5 \text{ XOR } 2 = 101 \text{ XOR } 010 = 111 = 7 \rightarrow S_1$

#### 4. Se extrage din această stivă un număr de obiecte a.î. restul stivelor să formeze o stare pară; nr de obiecte care rămân pe stivă este dat de XOR-ul între stiva respectivă și suma Nim

▪  $S_1 \text{ XOR } S_{Nim} = 3 \text{ XOR } 2 = 011 \text{ XOR } 010 = 001 = 1 \rightarrow$  se extrag 2 ( $= 3 - 1$ ) obiecte de pe stiva 1

#### 5. Se reia pasul 1

# Jocurile și căutarea – spațiul de căutare

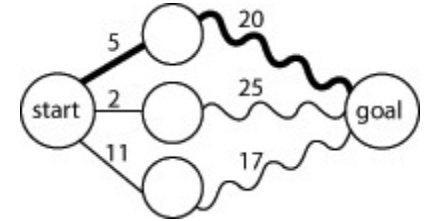


## Strategii de joc → Programare dinamică (PD)

### □ Aspecte teoretice

- Pași în rezolvarea unei probleme cu PD:
  - Descompunerea problemei în sub-probleme
  - Rezolvarea sub-problemelor
  - Combinarea sub-soluțiilor pentru obținerea soluției finale
- Pași în rezolvarea unui joc cu PD:
  - Descompunerea jocului în sub-jocuri
  - Găsirea unei strategii perfecte de câștig pentru fiecare sub-joc
  - Combinarea sub-strategiilor pentru obținerea strategiei finale

# Jocurile și căutarea—spațiul de căutare

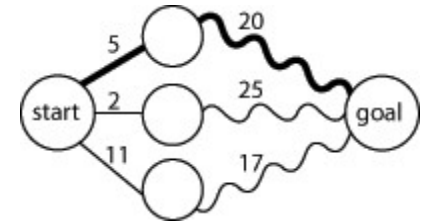


## Strategii de joc → Programare dinamică (PD)

### □ Aspecte teoretice

- Cum se rezolvă un joc cu programare dinamică?
  - descompunerea jocului în sub-jocuri  $J_h$  (sub-jocuri de pe nivelul cel mai de jos) și
    - etichetarea fiecărui joc  $J_h$  cu T (*true*) sau F (*false*) în funcție de posibilitatea jucătorului (care trebuie să mute din acea poziție) de a avea strategie sigură de câștig
  - un sub-joc  $J_k$  de pe un nivel interior ( $k < h$ ) va fi etichetat cu:
    - T, dacă există cel puțin un sub-joc  $J_i$ ,  $k < i$ , etichetat cu F, în care poate fi transformat jocul  $J_k$
    - F, dacă toate sub-jocurile  $J_i$ ,  $k < i$ , în care se poate ajunge printr-o mutare din jocul  $J_k$  sunt etichetate cu T

# Jocurile și căutarea—spațiul de căutare



Strategii de joc → Programare dinamică (PD)

## ■ Exemplu - Jocul *șirului de căsuțe*

### ■ Se dă:

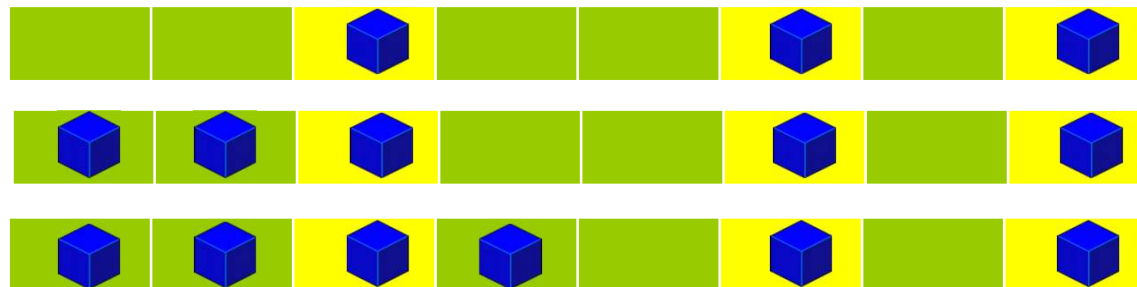
- În cele  $n$  căsuțe ale unui șir se află amplasate cuburi (maxim un cub în fiecare căsuță). Scopul jocului este umplerea tuturor căsuțelor cu cuburi. Alternativ, doi jucători A și B umplu cu cuburi (complet sau parțial) cel mai din stânga șir de căsuțe libere. Jucătorul care nu mai poate muta pierde. Jucătorul A mută primul.

### ■ Se cere:

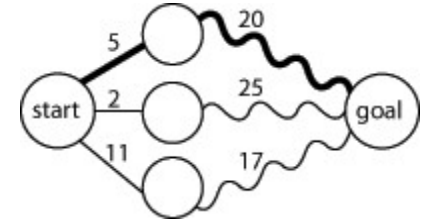
- Să se determine dacă jucătorul A are strategie sigură de câștig. În caz afirmativ, să se programeze mutările lui A, cele ale lui B fiind citite de la tastatură.

### ■ Exemplu

- $n = 8$



# Jocurile și căutarea—spațiul de căutare



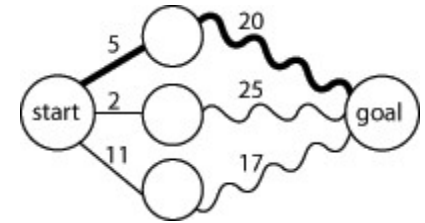
Strategii de joc → Programare dinamică (PD)

□ Exemplu - Jocul *șirului de căsuțe*

■ Soluție:



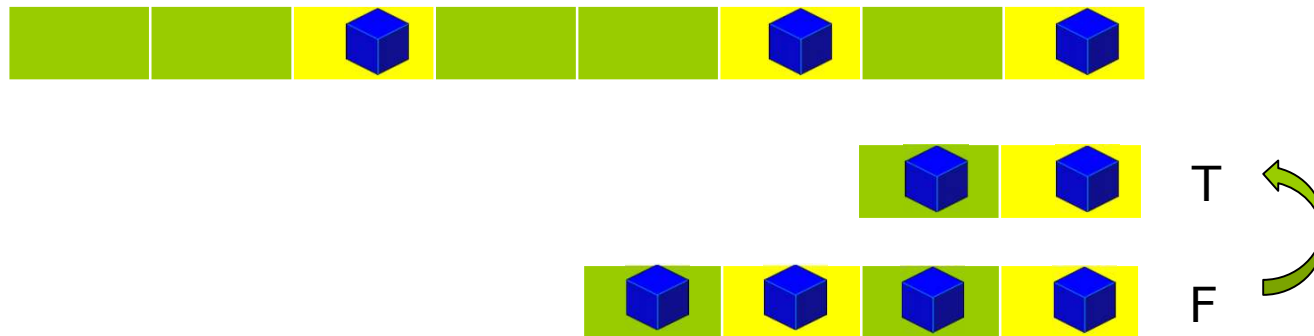
# Jocurile și căutarea—spațiul de căutare



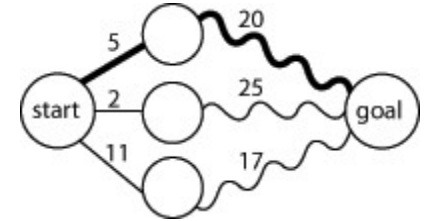
Strategii de joc → Programare dinamică (PD)

□ Exemplu - Jocul *șirului de căsuțe*

■ Soluție:



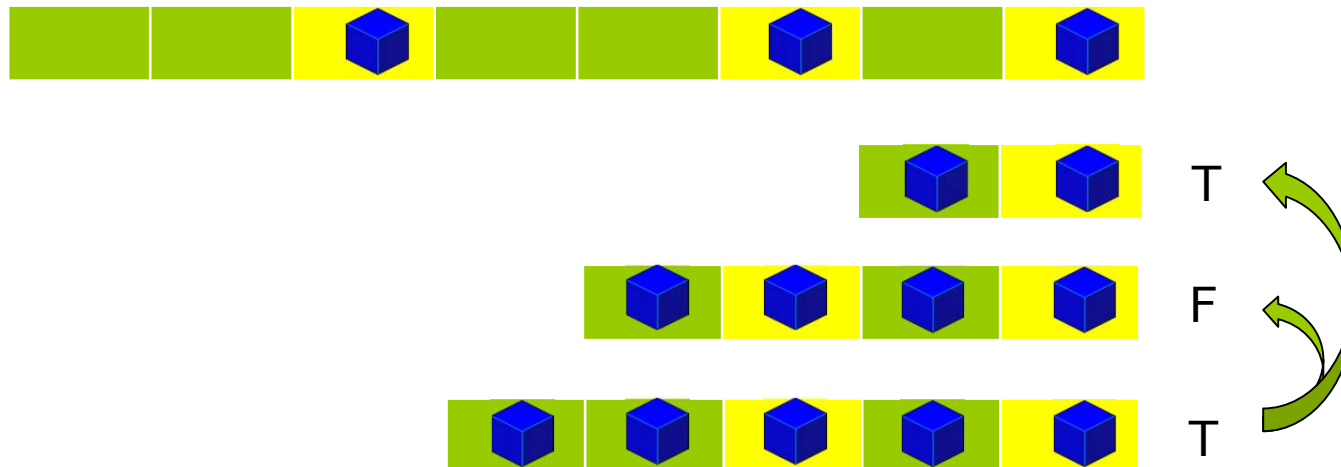
# Jocurile și căutarea—spațiul de căutare



Strategii de joc → Programare dinamică (PD)

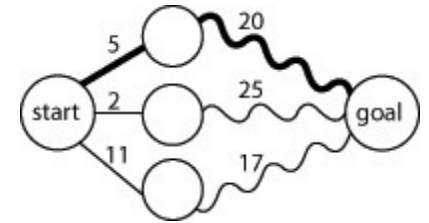
□ Exemplu - Jocul *șirului de căsuțe*

■ Soluție:





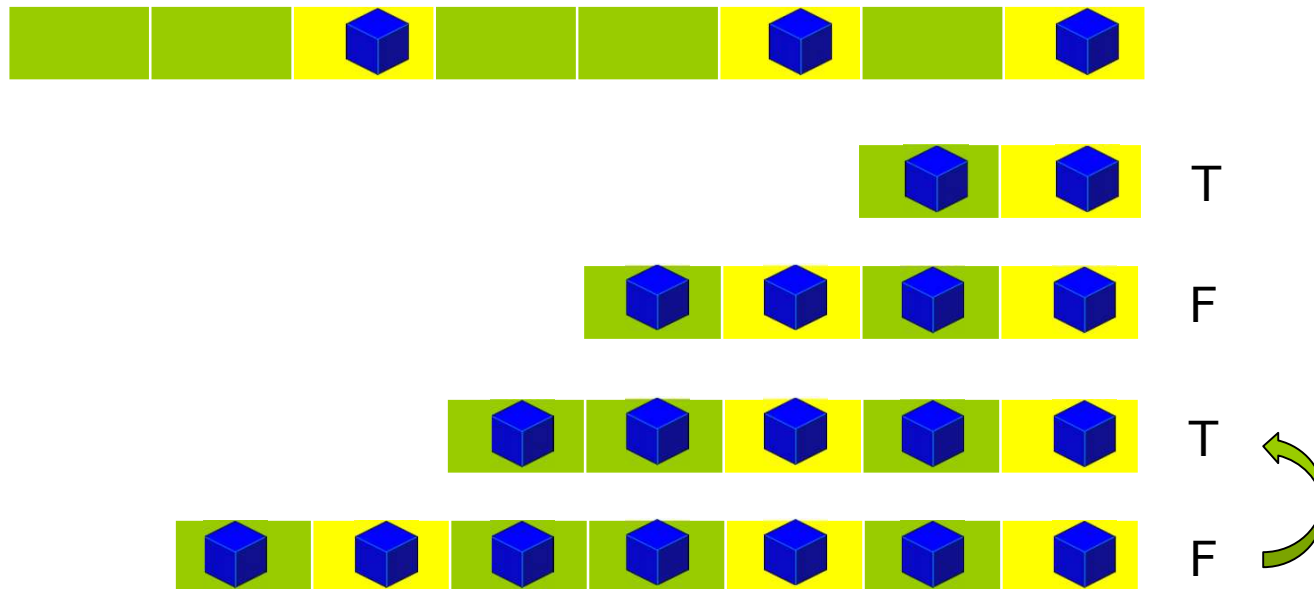
# Jocurile și căutarea—spațiul de căutare



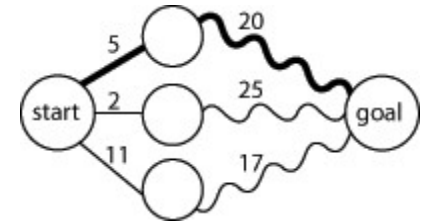
Strategii de joc → Programare dinamică (PD)

□ Exemplu - Jocul *șirului de căsuțe*

■ Soluție:



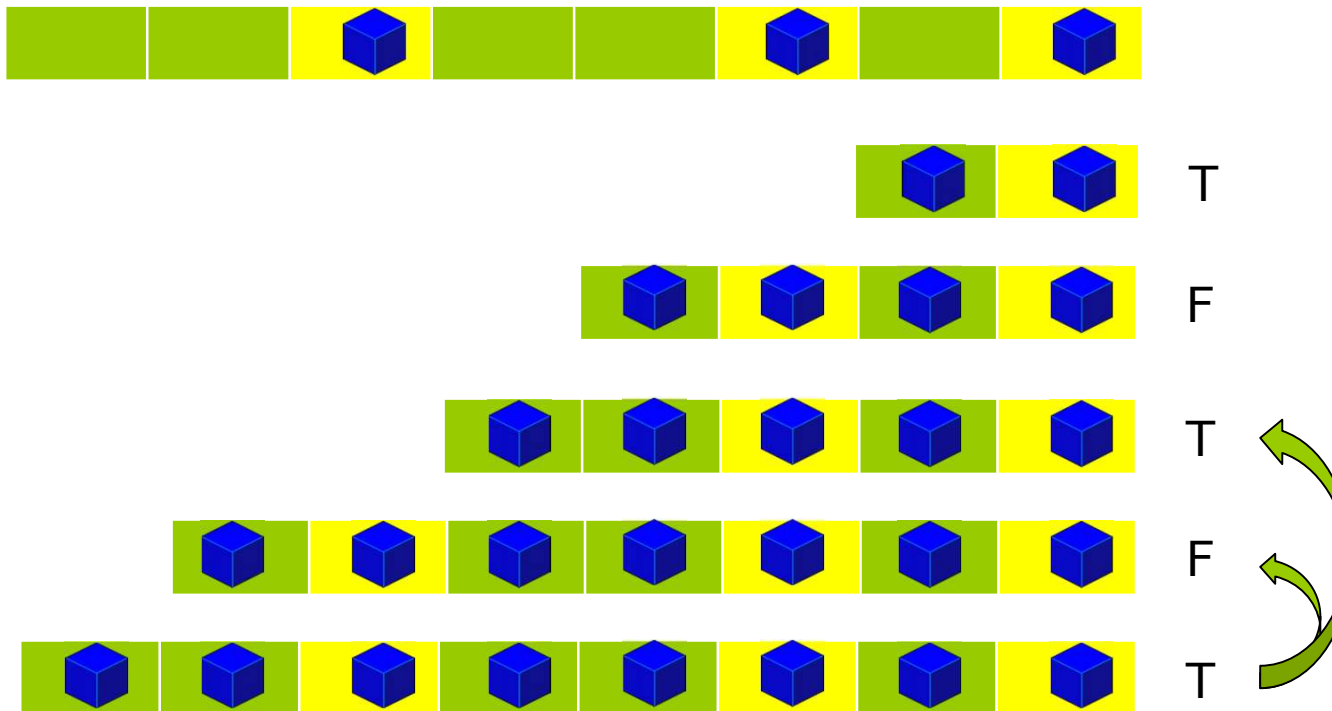
# Jocurile și căutarea—spațiul de căutare



Strategii de joc → Programare dinamică (PD)

□ Exemplu - Jocul *șirului de căsuțe*

■ Soluție:



# Jocurile și căutarea – spațiul de căutare



## □ Strategia de joc

### ■ Pas cu pas

- Ex.: XO, Dame, Șah
- Algoritmi – pot lucra cu structuri:

#### ■ Liniare

- Strategia simetriei
- Strategia perechilor
- Strategia parității
- Programare dinamică
- Alte strategii

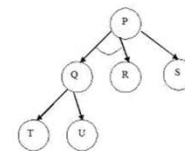
#### ■ **Arborescente**

- **Arbori AndOr**
- MiniMax (cu tăieturi Alpha-Beta)

### ■ Completă

- Ex.: ieșirea dintr-un labirint, deplasarea pe o hartă între 2 locații date
- Algoritmi pot să identifice
  - Un drum optim de la o locație la alta
  - O succesiune de acțiuni care să deplaseze jucătorul de la o locație la alta

# Jocurile și căutarea—spațiul de căutare

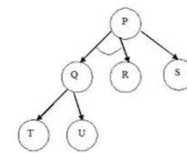


## Strategii de joc - Arbori And-Or (AAO)

### □ Aspecte teoretice

- **Reprezentarea spațiului de căutare** cu ajutorul AAO pentru un joc cu 2 jucători:
  - Partea (nodul) OR
    - → selectarea unei mutări pentru jucătorul curent A
    - → există o posibilitate (mutare) pentru jucătorul A să ajungă într-un nod AND
  - Partea (nodul) AND
    - → considerarea tuturor mutărilor posibile ale adversarului (jucătorul B)
    - → prin orice mutare jucătorul B ajunge într-un nod OR
  - Astfel:
    - Rădăcina → problema jucătorului câștigător (care începe jocul din starea inițială)
    - Mutările posibile ale primului jucător (A) se reunesc prin disjuncție (OR)
    - Mutările posibile ale celui de-al doilea jucător (B) se reunesc prin conjuncție (AND)
  - Jocul poate fi câștigat dacă începe cu un nod OR
- **Dificultăți:**
  - Arbori foarte foarte mari

# Jocurile și căutarea—spațiul de căutare



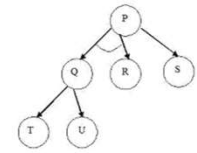
## Strategii de joc - Arbori And-Or (AAO)

### □ Aspecte teoretice

#### ■ Pași în rezolvarea unui joc cu AAO:

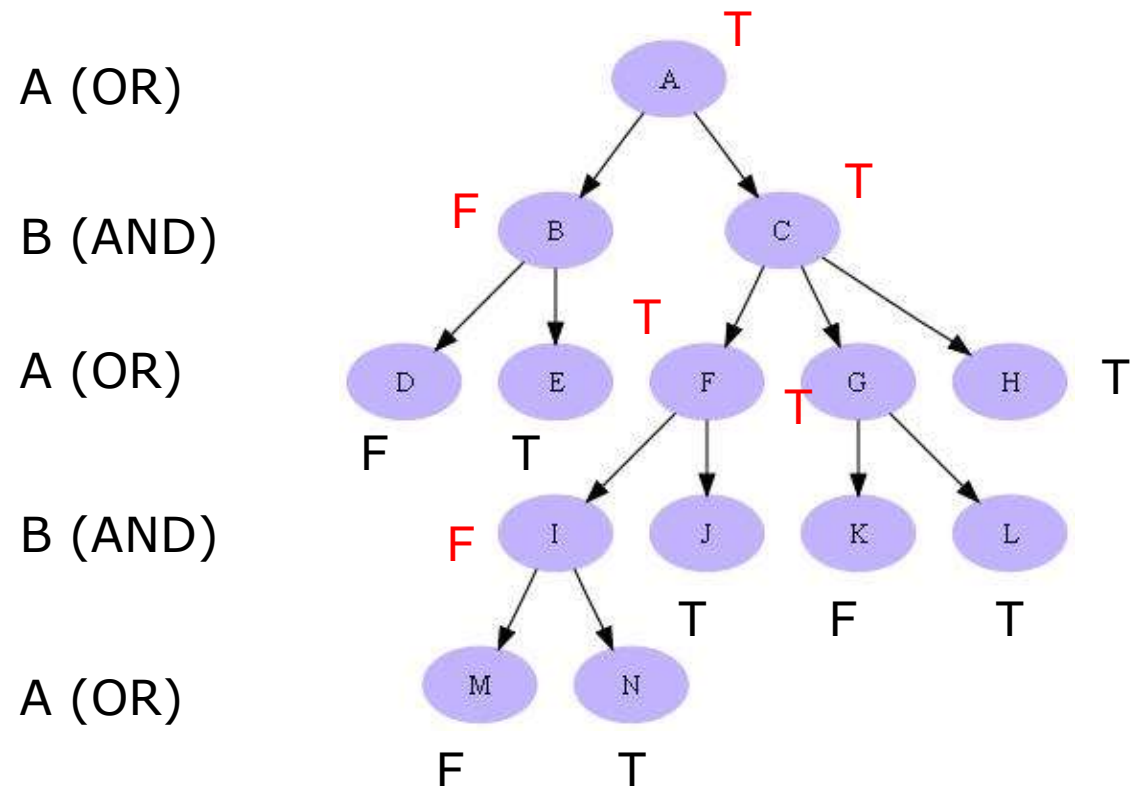
- Descompunerea jocului în sub-jocuri
  - noduri pe mai multe nivele,
  - nodurile de pe un nivel corespunzând mutărilor posibile ale unui jucător
- Găsirea unei strategii perfecte de câștig pentru fiecare sub-joc (nod)
  - Etichetarea nodurilor cu T sau F în funcție de posibilitatea jucătorului A de a avea o strategie sigură de câștig pentru acel sub-joc
  - Reguli de etichetare:
    - Frunzele se etichetează cu T sau F în funcție de configurația jocului
    - Nodurile interne se etichetează:
      - Pentru jucătorul A, cu T dacă **cel puțin un nod** fiu a fost etichetat cu T – regula OR
      - Pentru jucătorul B, cu T dacă **toate nodurile** fiu au fost etichetat cu T – regula AND
- Combinarea sub-strategiilor pentru obținerea strategiei finale

# Jocurile și căutarea—spațiul de căutare

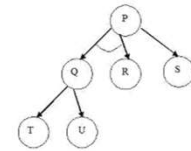


## Strategii de joc - Arbori And-Or (AAO)

### □ Exemplu



# Jocurile și căutarea—spațiul de căutare

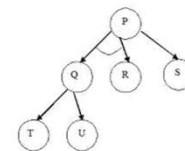


## Strategii de joc - Arbori And-Or (AAO)

### □ Algoritm

```
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;  
    }  
}
```

# Jocurile și căutarea—spațiul de căutare



## Strategii de joc - Arbori And-Or (AAO)

### □ Avantaje

- Pot fi aplicați pentru rezolvarea oricărui joc

### □ Dezavantaje

- Necesită multă memorie
- Necesită mult timp de calcul

### □ ➔ algoritmul mini-max



# Jocurile și căutarea – spațiul de căutare



## □ Strategia de joc

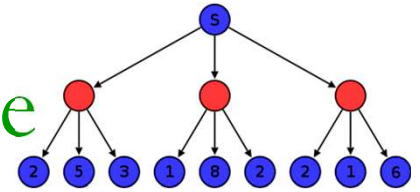
### ■ Pas cu pas

- Ex.: XO, Dame, Șah
- Algoritmi – pot lucra cu structuri:
  - Liniare
    - Strategia simetriei
    - Strategia perechilor
    - Strategia parității
    - Programare dinamică
    - Alte strategii
  - Arborescente
    - Arbori AndOr
    - **MiniMax** (cu tăieturi Alpha-Beta)

### ■ Completă

- Ex.: ieșirea dintr-un labirint, deplasarea pe o hartă între 2 locații date
- Algoritmi pot să identifice
  - Un drum optim de la o locație la alta
  - O succesiune de acțiuni care să deplaseze jucătorul de la o locație la alta

# Jocurile și căutarea – spațiul de căutare

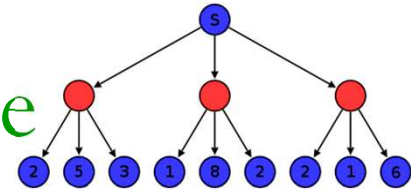


## Strategii de joc → MiniMax

### □ Aspecte teoretice

- Propus de John von Neuman în 1944
- Ideea de bază: maximizarea poziției unui jucător în timp ce poziția adversarului este minimizată
- Arborele de căutare constă în alternarea nivelelor pe care un jucător încearcă să-și maximizeze câștigul cu nivelele pe care adversarul minimizează câștigul (primului jucător)
  - Primul jucător va încerca să-și maximizeze câștigul (jucătorul MAX → mută primul)
  - Al doilea jucător va încerca să minimizeze câștigul primului jucător (jucătorul MIN → adversarul)
- Identificarea celor mai bune mutări
  - Se construiește arborele tuturor mutărilor posibile (fiecare mutare se aplică unei stări a jocului)
  - Se evaluează frunzele (care jucător câștigă)
  - Se propagă (în sus în arbore) câștigurile
  - Explorarea arborelui este de tip *Depth first search*
- Generarea tuturor mutărilor
  - Posibilă doar în jocuri simple (ex. Tic-Tac-Toe)
  - Imposibilă (cu resurse limitate) pentru jocuri complexe
    - reținând minimele în MIN
    - reținând maximele în MAX

# Jocurile și căutarea – spațiul de căutare



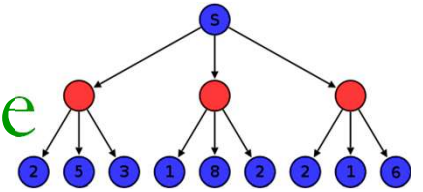
## Strategii de joc → MiniMax

### □ Algoritm

- Se construiește arborele corespunzător tuturor mutărilor
- Se evaluează frunzele
- Cât timp se mai poate alege un nod cu toți descendenții evaluați
  - Se alege un nod
  - Dacă nodul este de pe nivel Min, el va fi evaluat la cea mai mică valoare a unui descendent
  - Dacă nodul este de pe nivel Max, el va fi evaluat la cea mai mare valoare a unui descendent
- Se returnează valoarea nodului rădăcină (nod de pe nivel Max)

```
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;
    }
}
```

# Jocurile și căutarea – spațiul de căutare



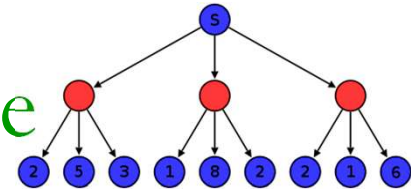
Strategii de joc → MiniMax

□ Exemplu – jocul *Tic-Tac-Toe*

■ Funcția de evaluare a unui nod:

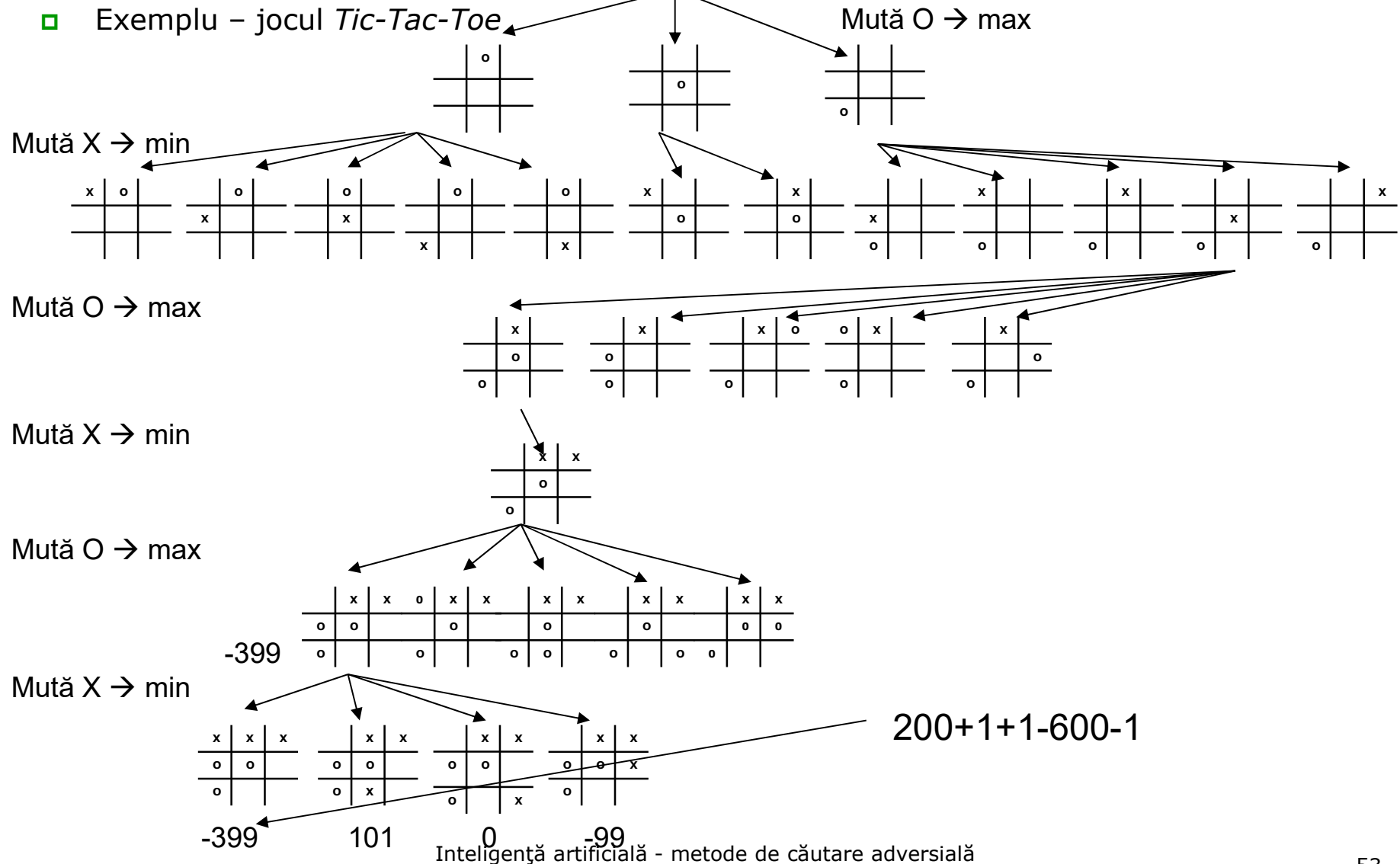
- Dacă există o linie aproape completă (cu 2 semne la fel) → 200 puncte
- Dacă există 2 linii aproape complete (cu 2 semne la fel) → 300 puncte
- O linie completă → 600 puncte
- Fiecare linie potențială → 1 punct
- Punctele jucătorului care urmează la mutare se adună
- Punctele celuilalt jucător se scad

# Jocurile și căutarea – spațiul de căutare

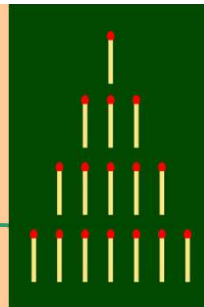


Strategii de joc → MiniMax

□ Exemplu – jocul *Tic-Tac-Toe*



# Jocurile și căutarea – spațiul de căutare



## Strategii de joc → MiniMax

### ■ Exemplu – *Jocul NIM*

#### ■ Se dă:

- N stive conțin fiecare  $p_i$  obiecte. 2 jucători A și B extrag, alternativ, dintr-o singură stivă oricâte obiecte. Jucătorul care execută ultima extragere câștigă jocul. Jucătorul A mută primul.

#### ■ Se cere:

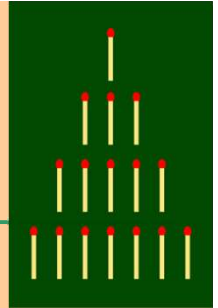
- Să se determine dacă jucătorul A are strategie sigură de câștig. În caz afirmativ, să se programeze mutările lui A, cele ale lui B fiind citite de la tastatură.

#### ■ Exemplu:

- 3 stive cu 1, 1 și, respectiv, 2 obiecte



# Jocurile și căutarea – spațiul de căutare



Strategii de joc → MiniMax

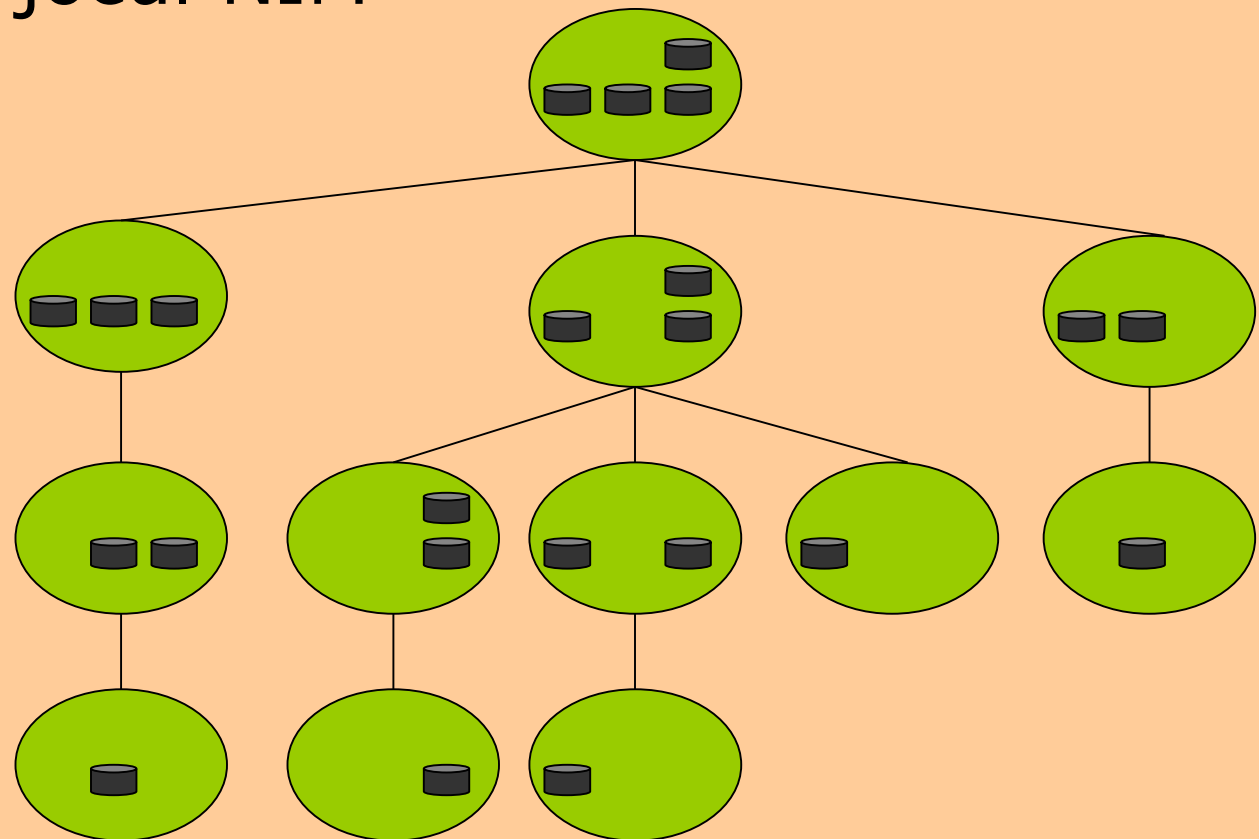
□ Exemplu – jocul NIM

Max

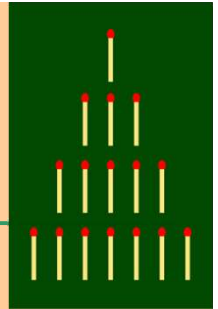
Min

Max

Min



# Jocurile și căutarea – spațiul de căutare



Strategii de joc → MiniMax

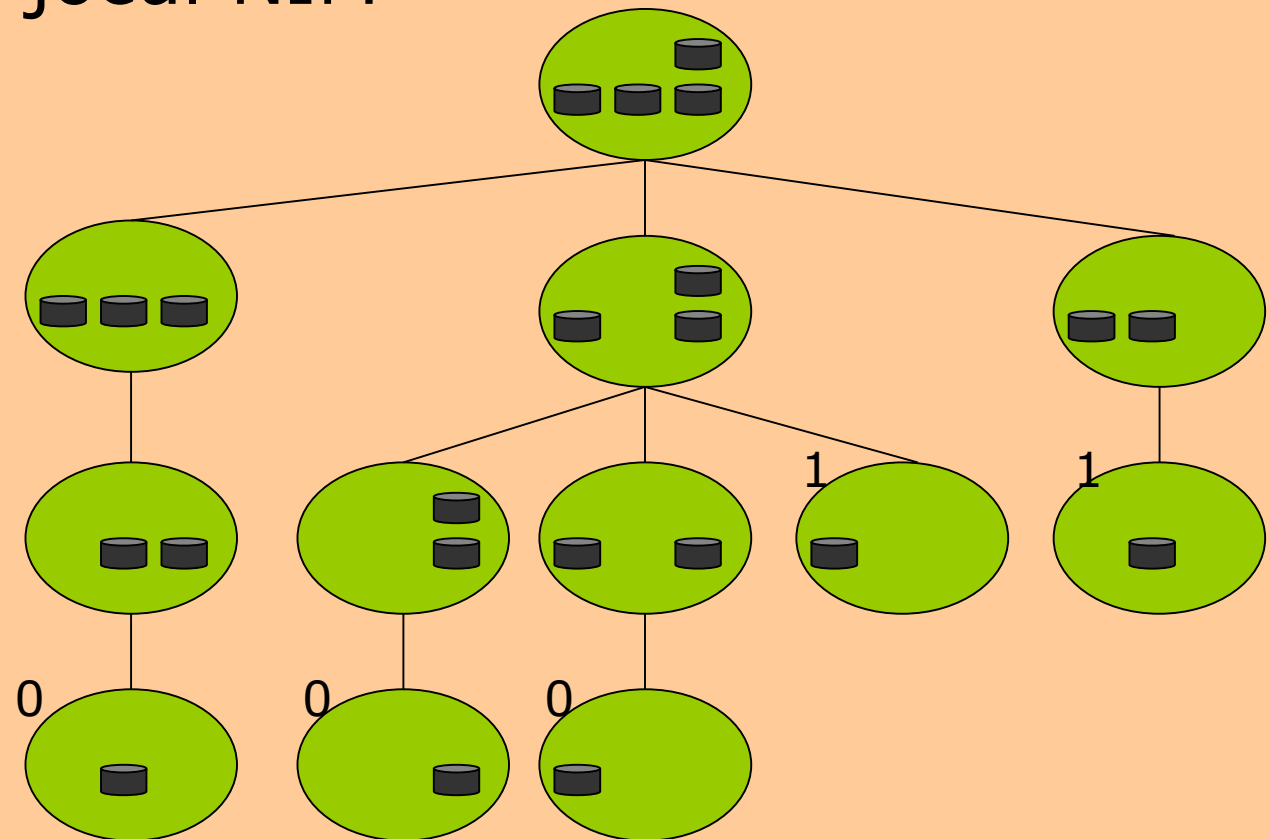
□ Exemplu – jocul NIM

Max

Min

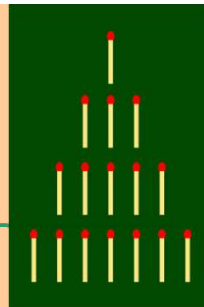
Max

Min





# Jocurile și căutarea – spațiul de căutare



Strategii de joc → MiniMax

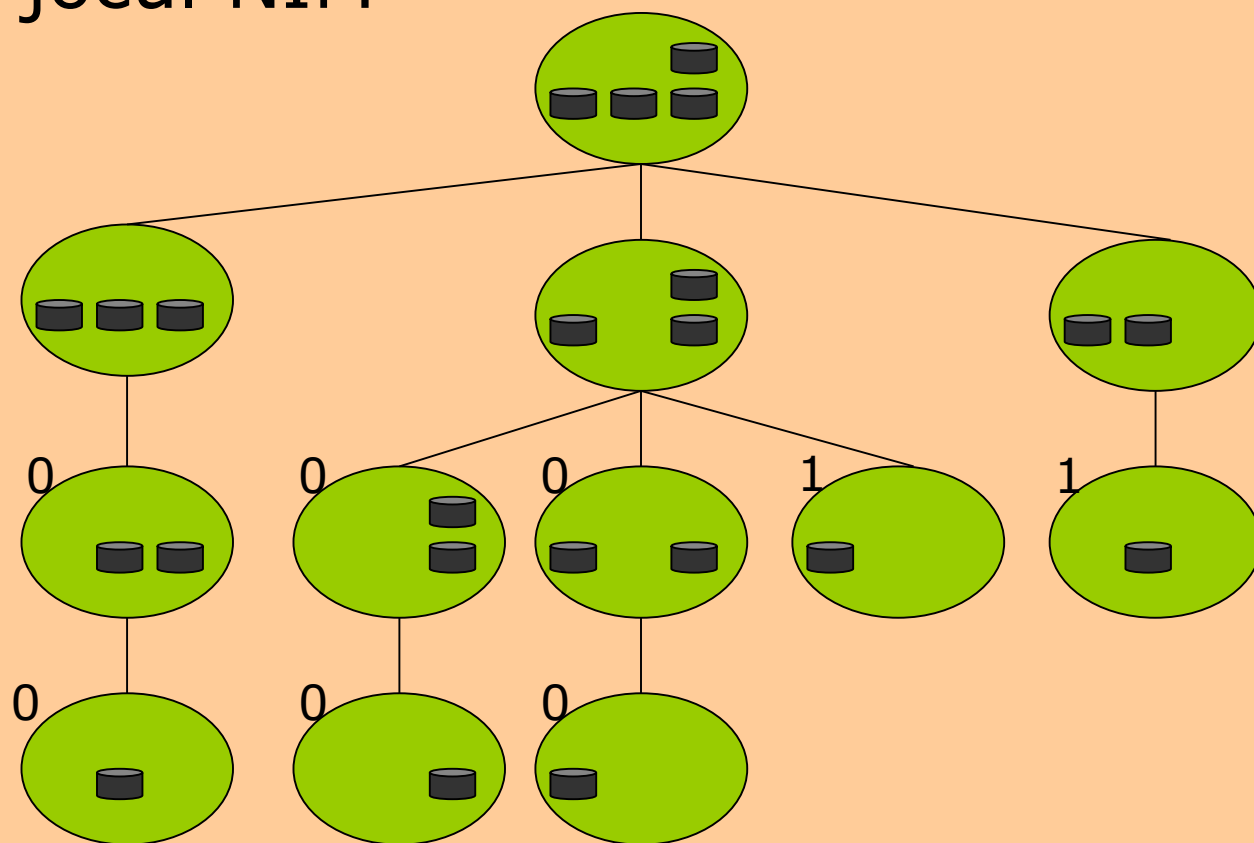
□ Exemplu – jocul NIM

Max

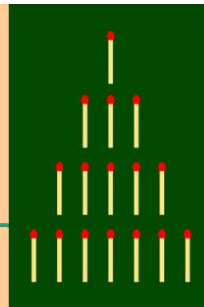
Min

Max

Min



# Jocurile și căutarea – spațiul de căutare



Strategii de joc → MiniMax

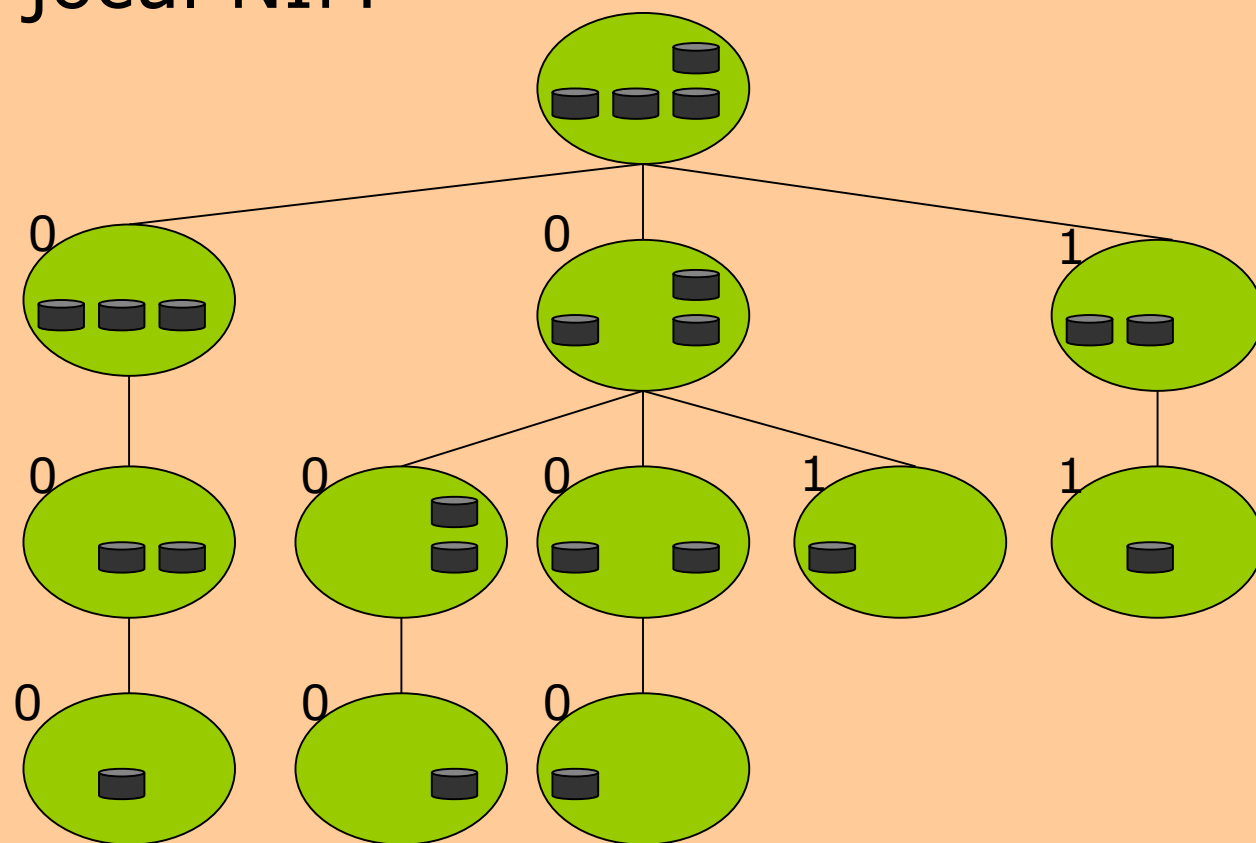
□ Exemplu – jocul NIM

Max

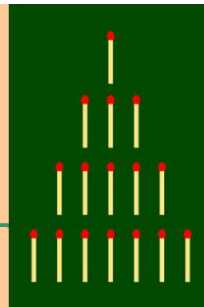
Min

Max

Min



# Jocurile și căutarea – spațiul de căutare



Strategii de joc → MiniMax

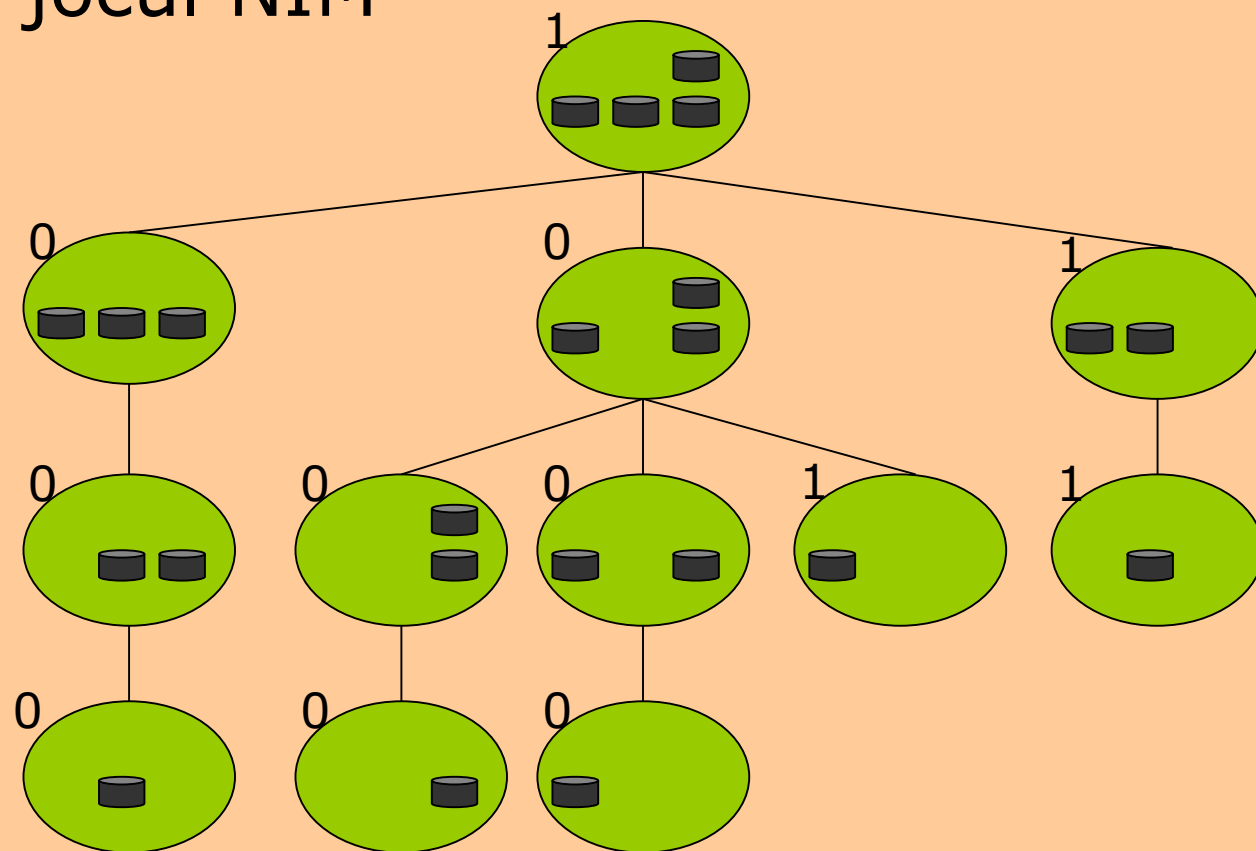
□ Exemplu – jocul NIM

Max

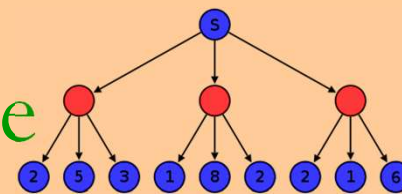
Min

Max

Min



# Jocurile și căutarea – spațiul de căutare

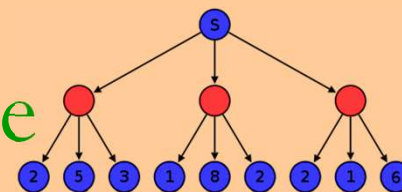


## Strategii de joc → MiniMax

### □ Dificultăți

- Nu explorează întregul arbore
  - Căutare depth-first
- La începutul jocului se fixează:
  - O adâncime maximă
    - Care este adâncimea optimă?
      - Adâncime mare → căutare lungă
      - Adâncime mică → anumite drumuri pot fi ratate (sacrificii timpurii pentru câștiguri târzii)
  - O funcție de evaluare
    - Fiecare nod (stare) trebuie evaluat(ă)
    - Cum? → folosind euristici

# Jocurile și căutarea – spațiul de căutare



## Strategii de joc → MiniMax

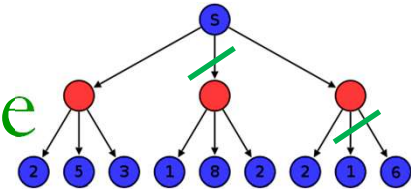
### □ Dezavantaje

- Doar 100 noduri sunt explorate într-o secundă
- Sah – timp mutare = 150s → 150 000 de poziții  
→ doar 3 - 4 mutări în avans

### □ Soluția

- Evitarea anumitor noduri prin retezarea unor ramuri (redundante) → retezare (reducere)  
alpha-beta → MiniMax cu retezare  $\alpha$ - $\beta$

# Jocurile și căutarea – spațiul de căutare



## Strategii de joc → MiniMax cu retezare $\alpha$ - $\beta$

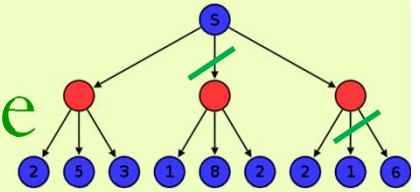
### □ Minimax

- Crează întreg arborele
- Se propagă valorile de jos în sus

### □ MiniMax cu retezare $\alpha$ - $\beta$

- Crearea și propagarea în același timp
- Dacă se știe că un drum este rău, nu se mai consumă energie ca să se afle cât de rău este acel drum
  - Se pot elimina anumite evaluări inutile
  - Se pot elimina anumite expandări ale nodurilor

# Jocurile și căutarea – spațiul de căutare



## Strategii de joc → MiniMax cu retezare $\alpha$ - $\beta$

### □ Apecte teoretice

- Descoperit de McCarthy în 1956, dar publicat pentru prima dată într-un raport tehnic de la MIT

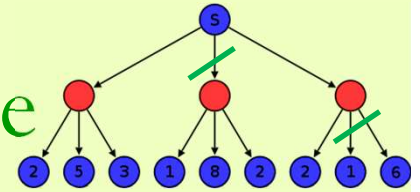
### ■ Ideea de bază

- Valoarea minimax a rădăcinii poate fi determinată fără examinarea tuturor nodurilor de pe frontiera căutării
- Similar algoritmului *branch and bound*

### ■ Denumirea $\alpha$ - $\beta$

- $\alpha$  - valoarea celei mai bune alegeri (cele mai mari) efectuate de-a lungul drumului urmat de MAX
  - dacă o valoare unui nod este mai slabă decât  $\alpha$  atunci MAX o va evita și va reteza sub-arborele cu rădăcina în acel nod
- $\beta$  - similar lui  $\alpha$ , dar pentru jucătorul MIN

# Jocurile și căutarea – spațiul de căutare



## Strategii de joc → MiniMax cu retezare $\alpha$ - $\beta$

### □ Aspecte teoretice

#### ■ $\alpha$

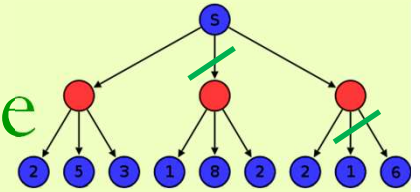
- valoarea cele mai bune (adică cea mai mare) alegeri găsită până la momentul curent în orice punct de-a lungul unui drum pentru MAX
- scorul minim pe care jucătorul MAX îl poate obține în mod garantat
- dacă un nod  $v$  este mai slab (mai mic) decât  $\alpha$ , MAX îl va evita prin eliminarea acelei ramuri →
  - Dacă s-a ajuns cu căutarea într-un nod MIN a cărui valoare  $\leq \alpha$  atunci descendenții aceluși nod nu mai merită explorați pentru că oricum MAX îi va ignora

#### ■ $\beta$

- cea mai mică valoare găsită la orice punct de-a lungul unui drum pentru MIN
- scorul minim pe care jucătorul MAX speră să-l obțină
- dacă un nod  $v$  este mai bun (mai mare) decât  $\beta$ , MIN îl va evita prin eliminarea acelei ramuri →
  - Dacă s-a ajuns cu căutarea într-un nod MAX a cărui valoare  $\geq \beta$  atunci descendenții aceluși nod nu mai merită explorați pentru că oricum MIN îi va ignora



# Jocurile și căutarea – spațiul de căutare

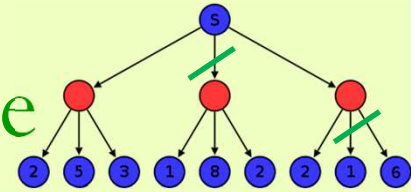


## Strategii de joc → MiniMax cu retezare $\alpha$ - $\beta$

### □ Aspecte teoretice

- Valoarea  $\alpha$  a unui nod
  - Inițial, scorul aceluia nod (dacă este frunză) sau  $-\infty$
  - Apoi,
    - Nod de tip MAX = cel mai mare scor al descendenților
    - Nod de tip MIN = valoarea  $\alpha$  a predecesorului
- Valoarea  $\beta$  a unui nod
  - Inițial, scorul aceluia nod (dacă este frunză) sau  $+\infty$
  - Apoi,
    - Nod de tip MIN = cel mai mic scor al descendenților
    - Nod de tip MAX = valoarea  $\beta$  a predecesorului
- Scorul unui nod
  - Nod de tip MAX valoarea  $\alpha$  finală
  - Nod de tip MIN valoarea  $\beta$  finală

# Jocurile și căutarea – spațiul de căutare



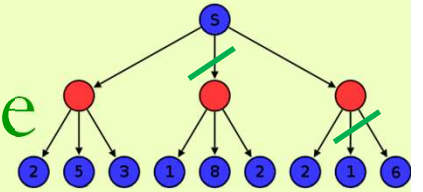
## Strategii de joc → MiniMax cu retezare $\alpha$ - $\beta$

### □ Algoritm

```

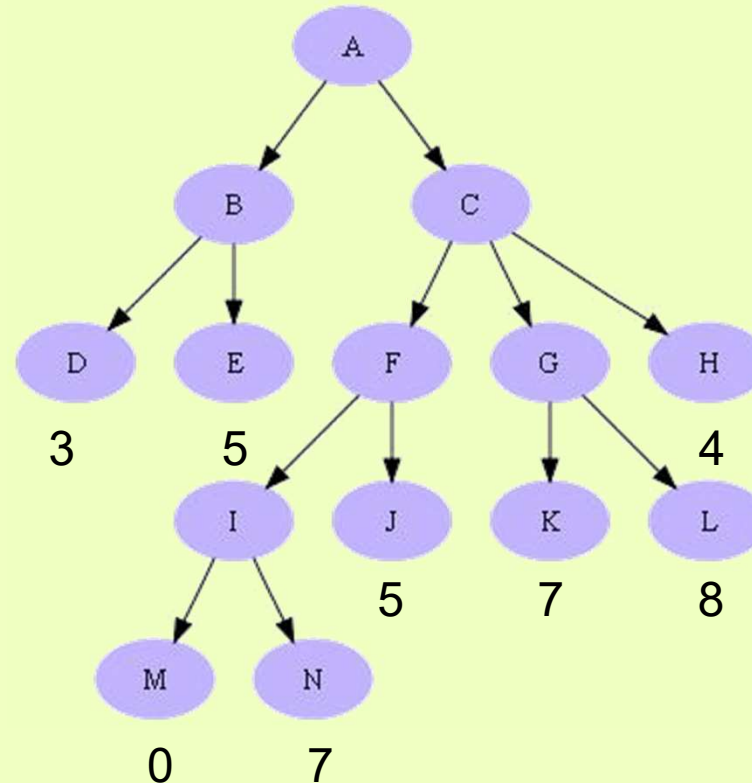
int backMiniMaxAB(Node N, int A, int B){
    Set Alpha value of N to A and Beta value of N to B;
    if N is a leaf
        return the estimated score of this leaf
    else{
        if N is a Min node
            for each child Ni of N {
                Val = backMiniMaxAB(Ni, Alpha of N, Beta of N);
                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;
    }
}
    
```

# Jocurile și căutarea – spațiul de căutare

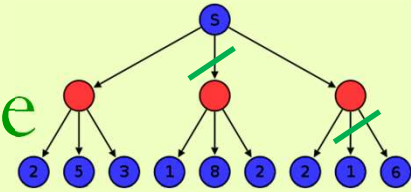


Strategii de joc → MiniMax cu retezare  $\alpha$ - $\beta$

□ Exemplu



# Jocurile și căutarea – spațiul de căutare



Strategii de joc → MiniMax cu retezare  $\alpha$ - $\beta$

□ Exemplu

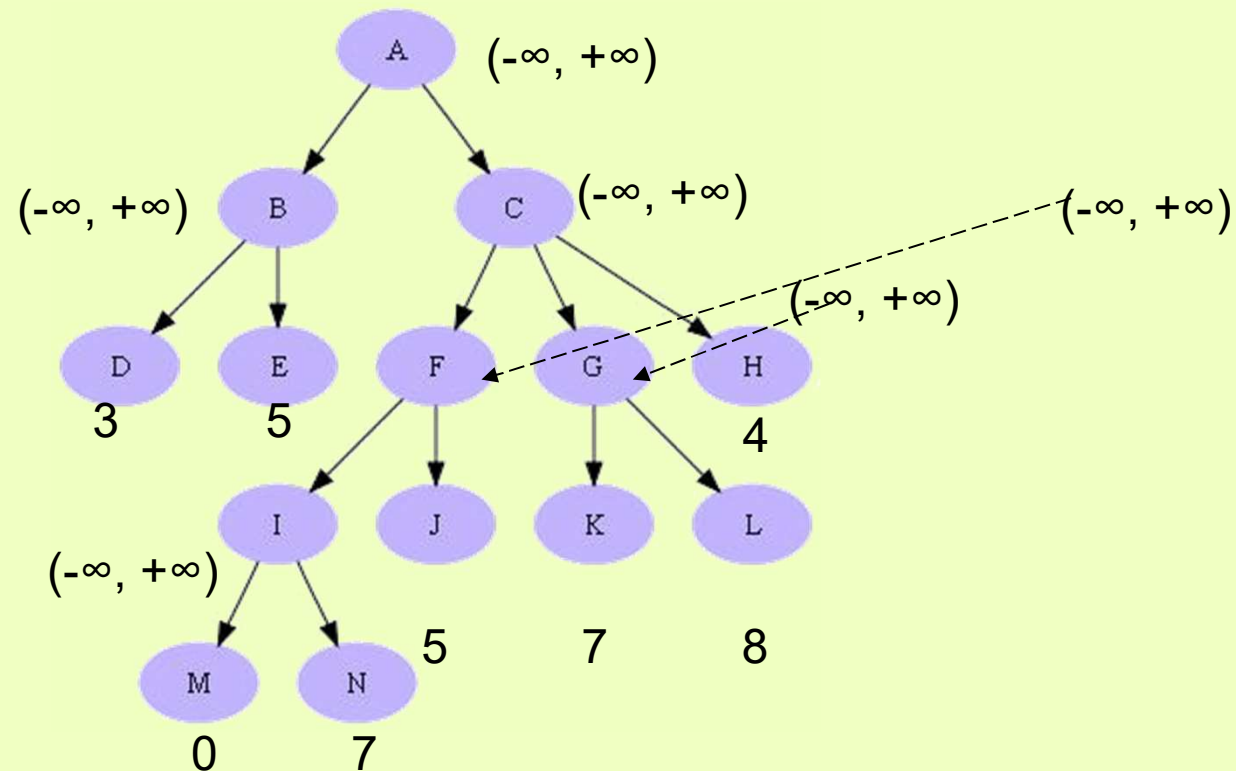
MAX

MIN

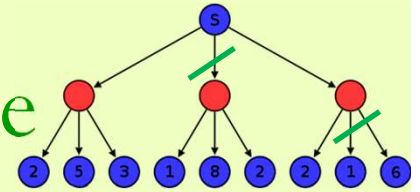
MAX

MIN

MAX



# Jocurile și căutarea – spațiul de căutare



Strategii de joc → MiniMax cu retezare  $\alpha$ - $\beta$

□ Exemplu

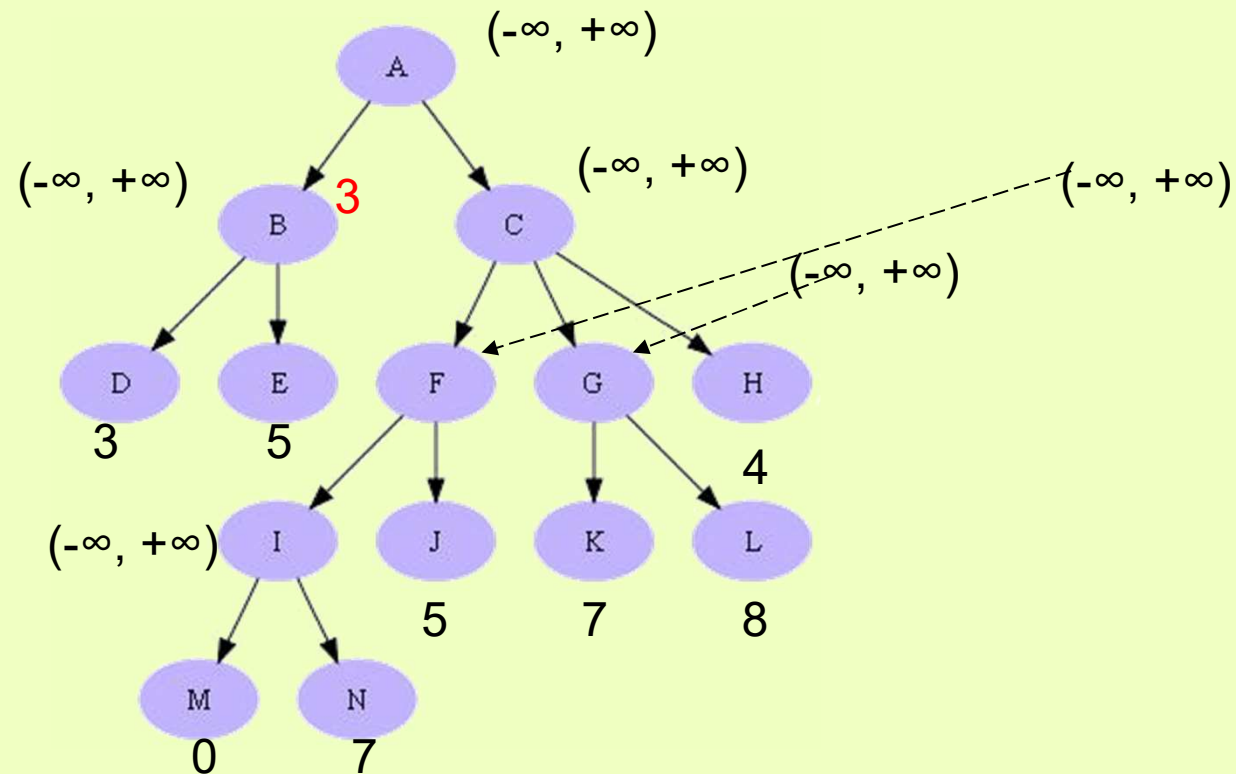
MAX( $\alpha$ )

MIN( $\beta$ )

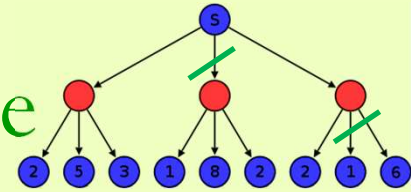
MAX( $\alpha$ )

MIN( $\beta$ )

MAX( $\alpha$ )



# Jocurile și căutarea – spațiul de căutare



Strategii de joc → MiniMax cu retezare  $\alpha$ - $\beta$

□ Exemplu

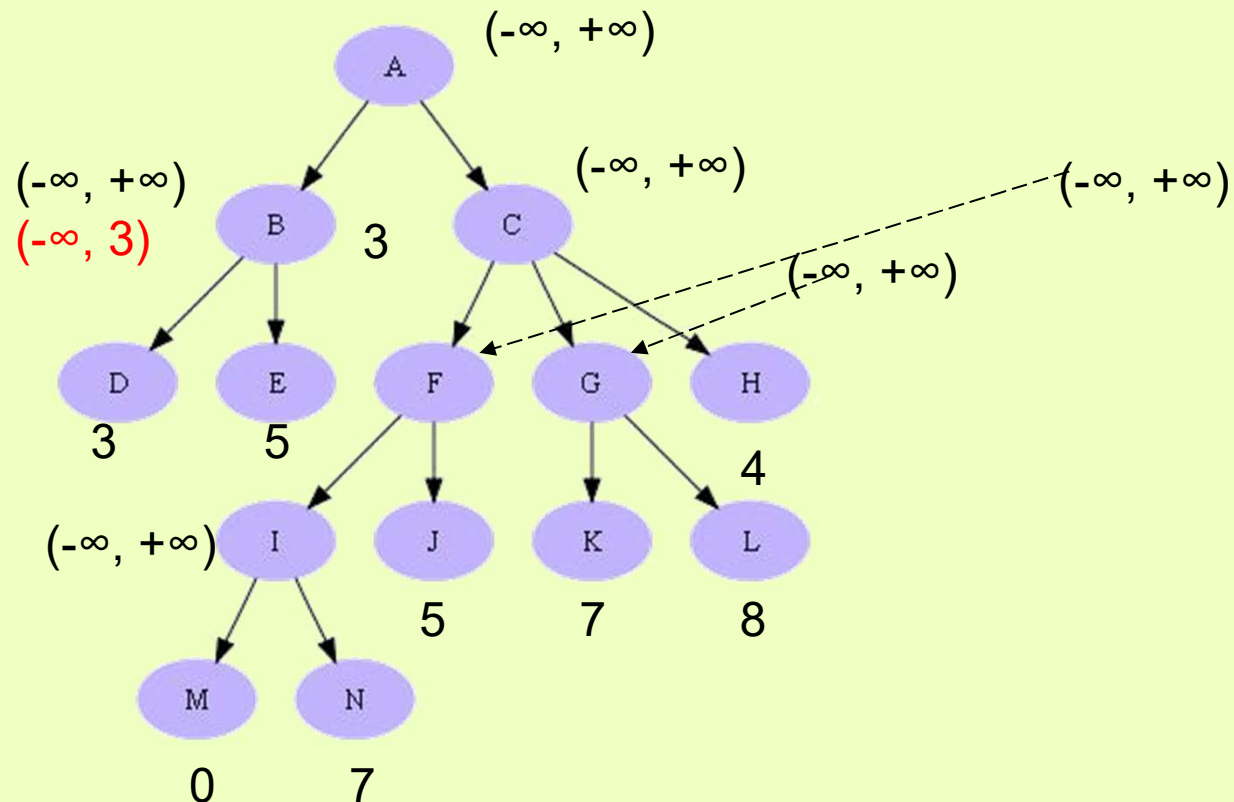
MAX( $\alpha$ )

MIN( $\beta$ )

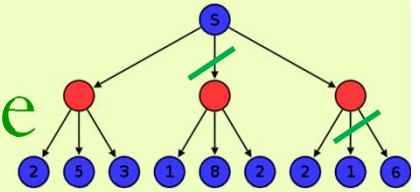
MAX( $\alpha$ )

MIN( $\beta$ )

MAX( $\alpha$ )



# Jocurile și căutarea – spațiul de căutare



Strategii de joc → MiniMax cu retezare  $\alpha$ - $\beta$

□ Exemplu

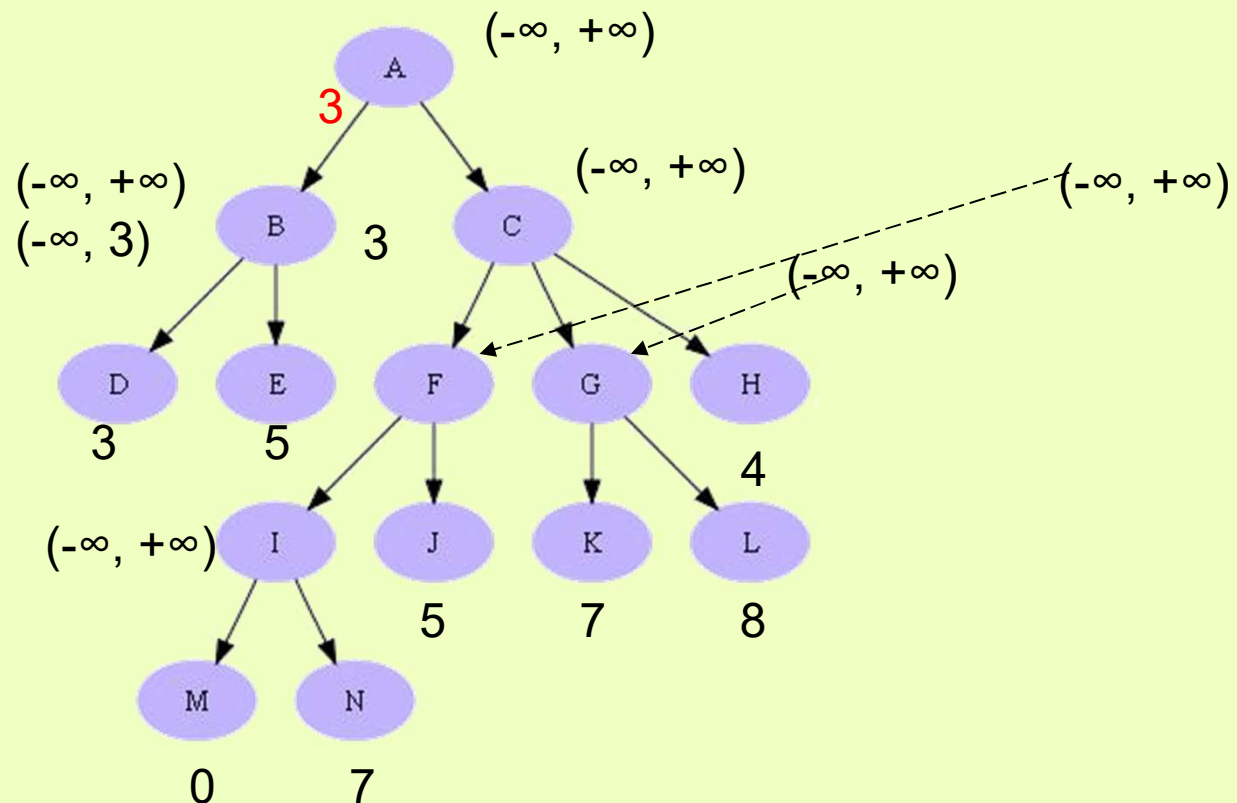
MAX( $\alpha$ )

MIN( $\beta$ )

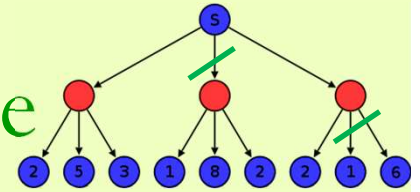
MAX( $\alpha$ )

MIN( $\beta$ )

MAX( $\alpha$ )



# Jocurile și căutarea – spațiul de căutare



Strategii de joc → MiniMax cu retezare  $\alpha$ - $\beta$

□ Exemplu

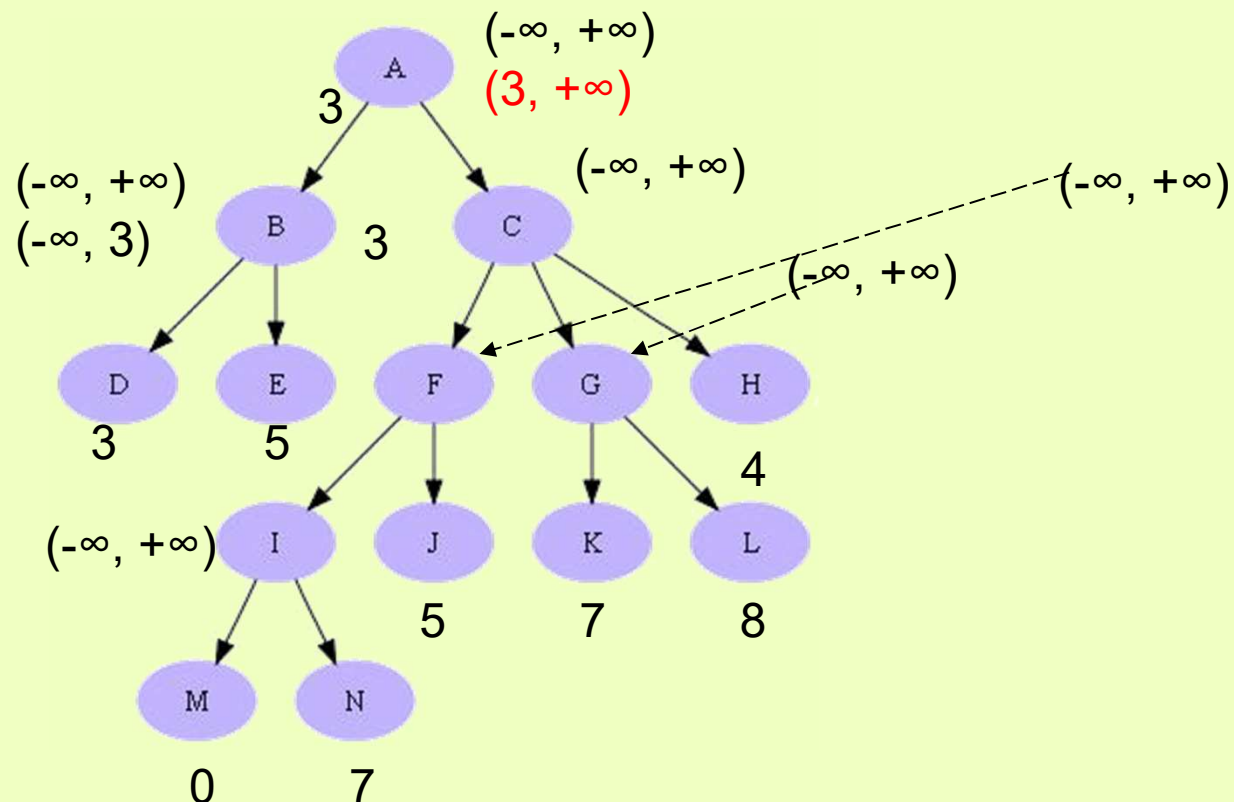
MAX( $\alpha$ )

MIN( $\beta$ )

MAX( $\alpha$ )

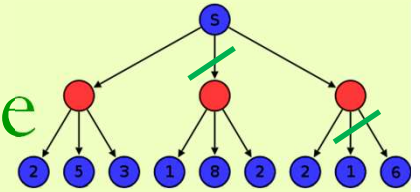
MIN( $\beta$ )

MAX( $\alpha$ )





# Jocurile și căutarea – spațiul de căutare



Strategii de joc → MiniMax cu retezare  $\alpha$ - $\beta$

□ Exemplu

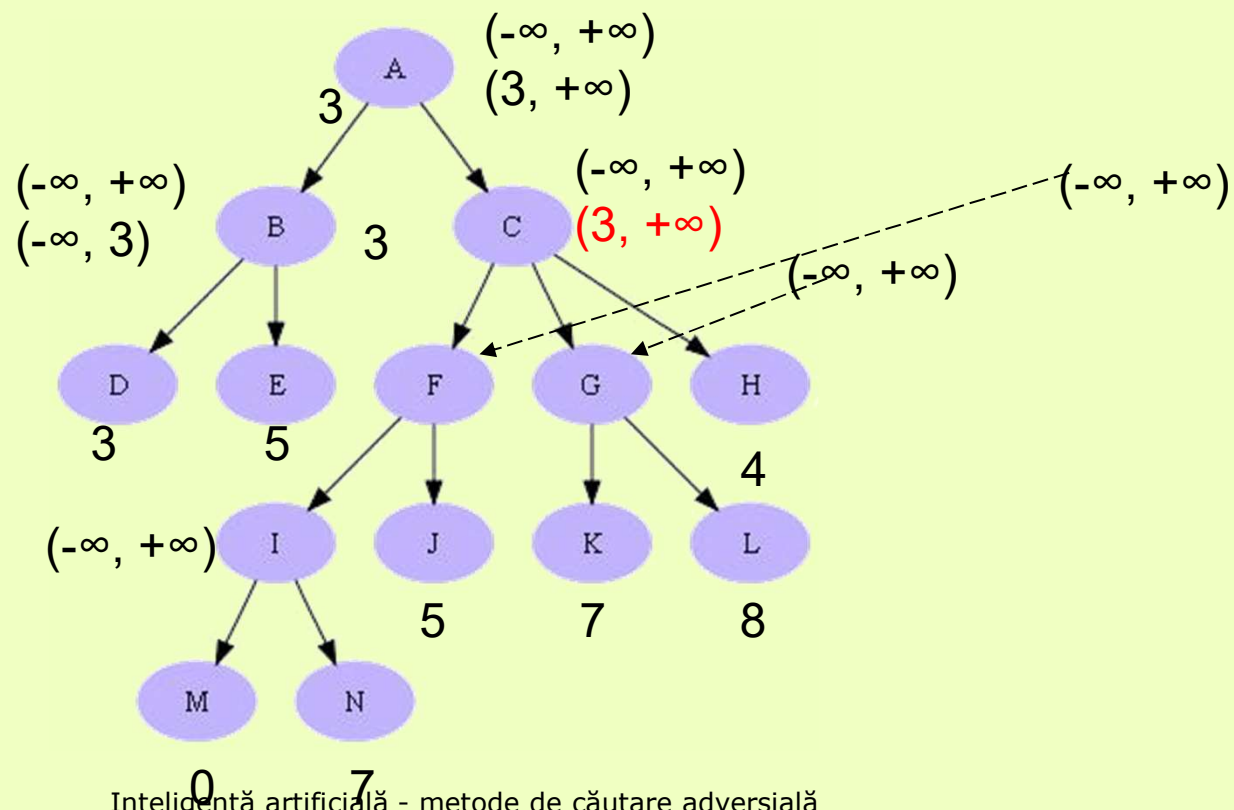
MAX( $\alpha$ )

MIN( $\beta$ )

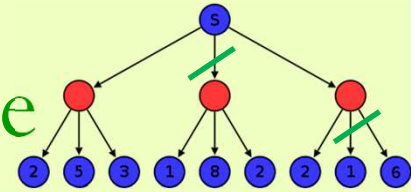
MAX( $\alpha$ )

MIN( $\beta$ )

MAX( $\alpha$ )



# Jocurile și căutarea – spațiul de căutare



Strategii de joc → MiniMax cu retezare  $\alpha$ - $\beta$

□ Exemplu

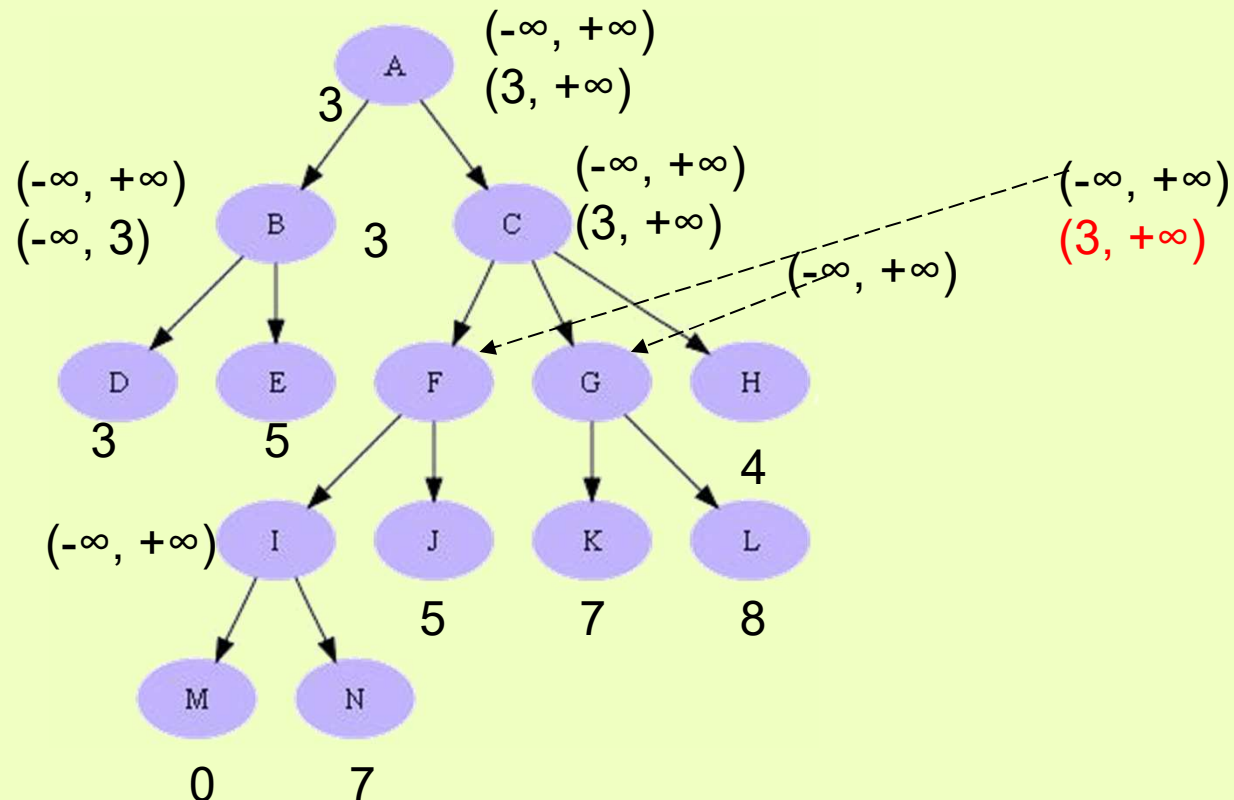
MAX( $\alpha$ )

MIN( $\beta$ )

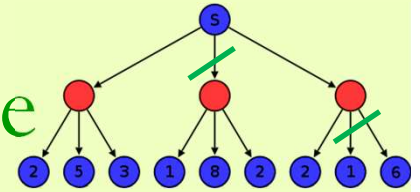
MAX( $\alpha$ )

MIN( $\beta$ )

MAX( $\alpha$ )



# Jocurile și căutarea – spațiul de căutare



Strategii de joc → MiniMax cu retezare  $\alpha$ - $\beta$

□ Exemplu

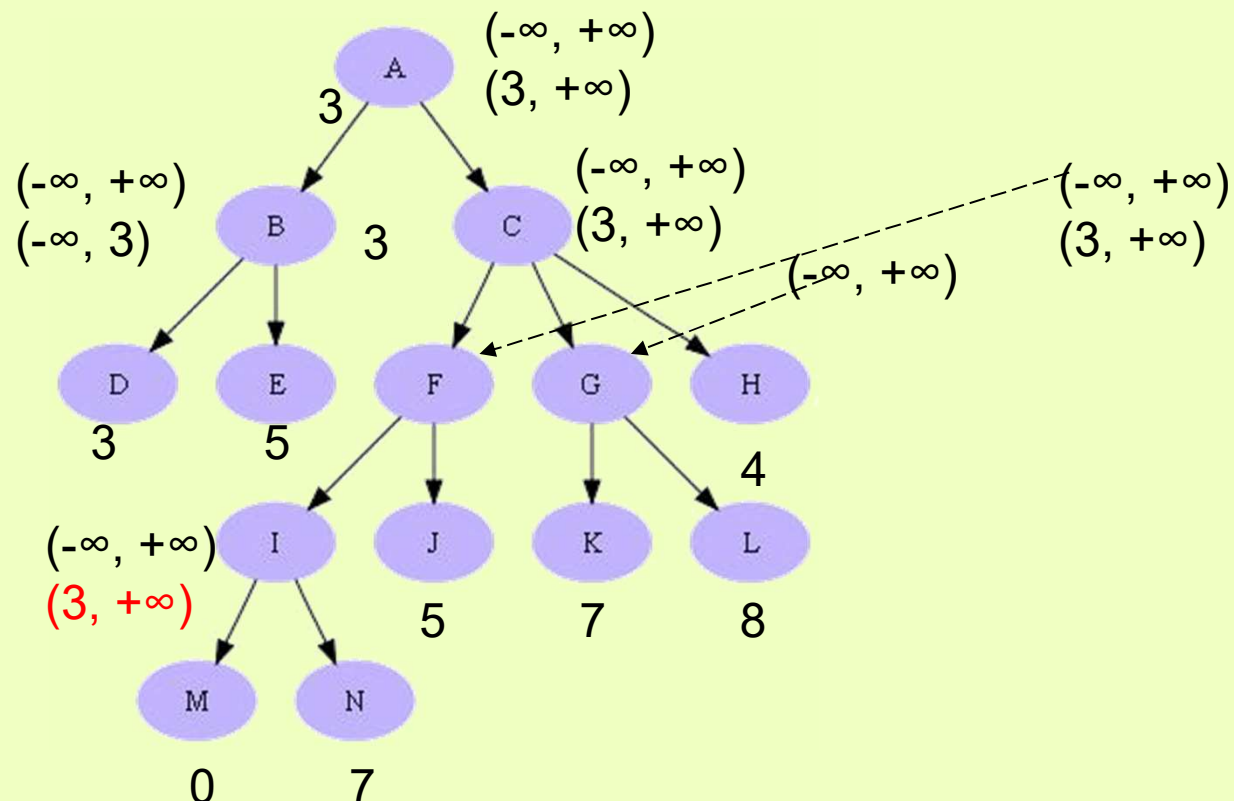
MAX( $\alpha$ )

MIN( $\beta$ )

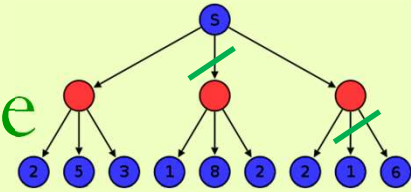
MAX( $\alpha$ )

MIN( $\beta$ )

MAX( $\alpha$ )



# Jocurile și căutarea – spațiul de căutare



Strategii de joc → MiniMax cu retezare  $\alpha$ - $\beta$

□ Exemplu

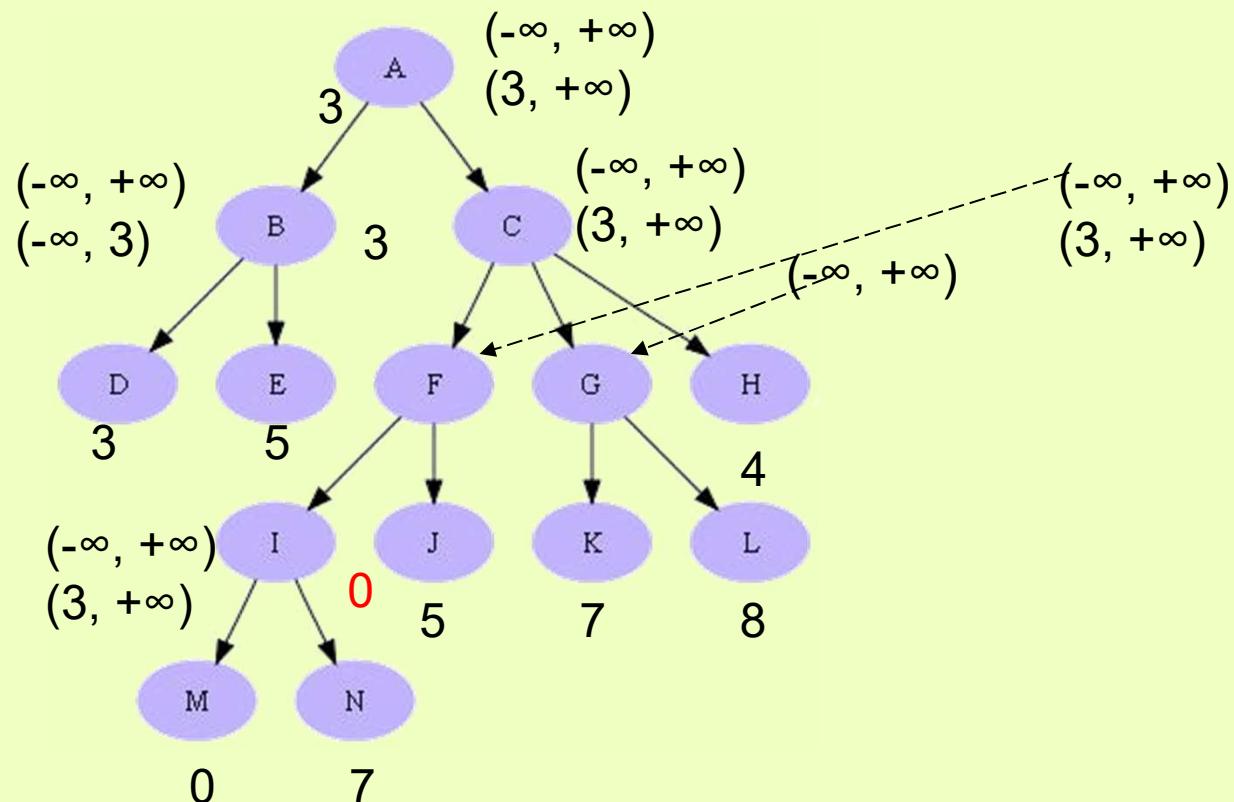
MAX( $\alpha$ )

MIN( $\beta$ )

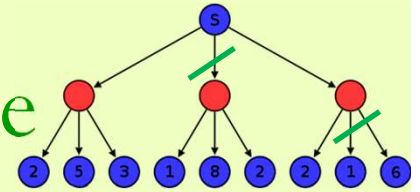
MAX( $\alpha$ )

MIN( $\beta$ )

MAX( $\alpha$ )



# Jocurile și căutarea – spațiul de căutare



Strategii de joc → MiniMax cu retezare  $\alpha$ - $\beta$

□ Exemplu

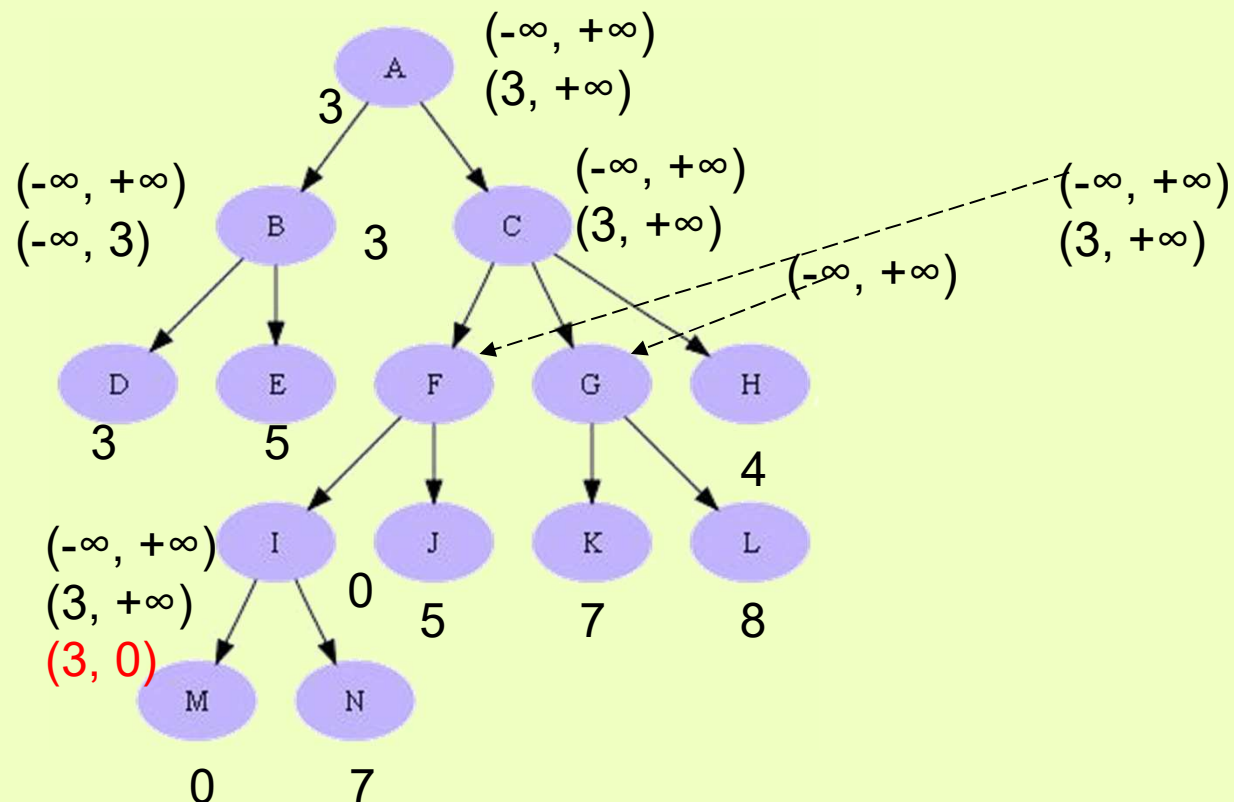
MAX( $\alpha$ )

MIN( $\beta$ )

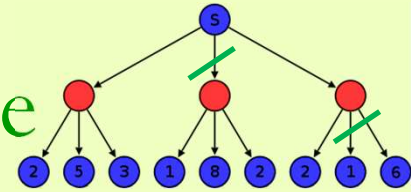
MAX( $\alpha$ )

MIN( $\beta$ )

MAX( $\alpha$ )



# Jocurile și căutarea – spațiul de căutare



Strategii de joc → MiniMax cu retezare  $\alpha$ - $\beta$

□ Exemplu

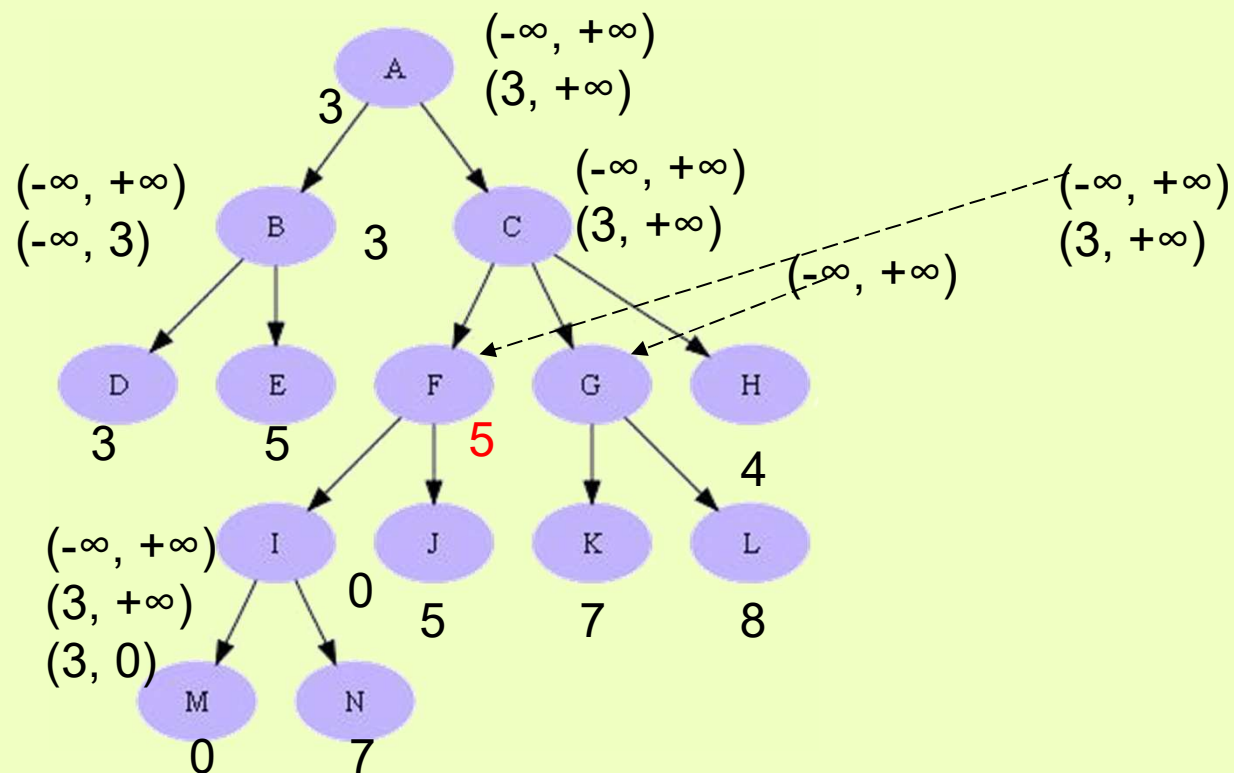
MAX( $\alpha$ )

MIN( $\beta$ )

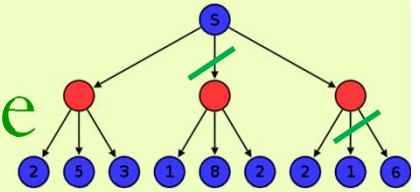
MAX( $\alpha$ )

MIN( $\beta$ )

MAX( $\alpha$ )



# Jocurile și căutarea – spațiul de căutare



Strategii de joc → MiniMax cu retezare  $\alpha$ - $\beta$

□ Exemplu

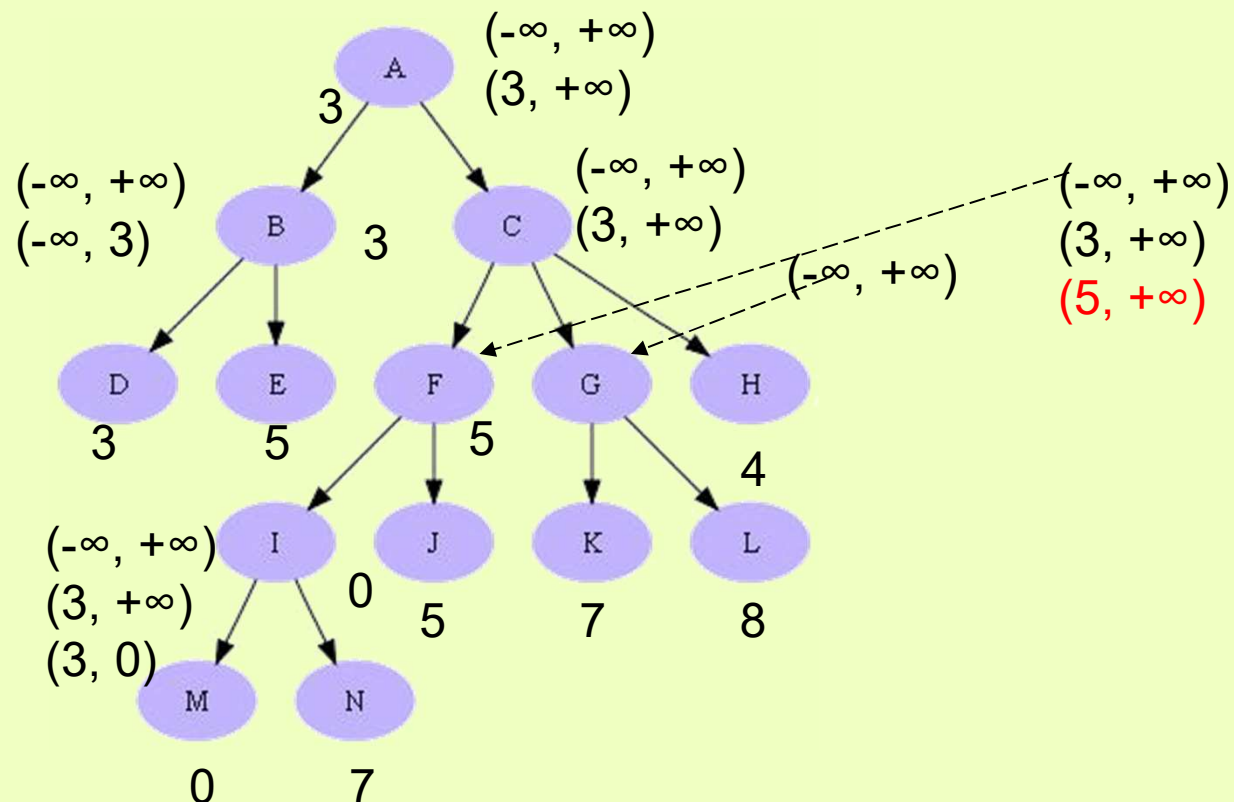
MAX( $\alpha$ )

MIN( $\beta$ )

MAX( $\alpha$ )

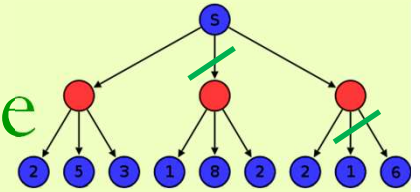
MIN( $\beta$ )

MAX( $\alpha$ )





# Jocurile și căutarea – spațiul de căutare



Strategii de joc → MiniMax cu retezare  $\alpha$ - $\beta$

□ Exemplu

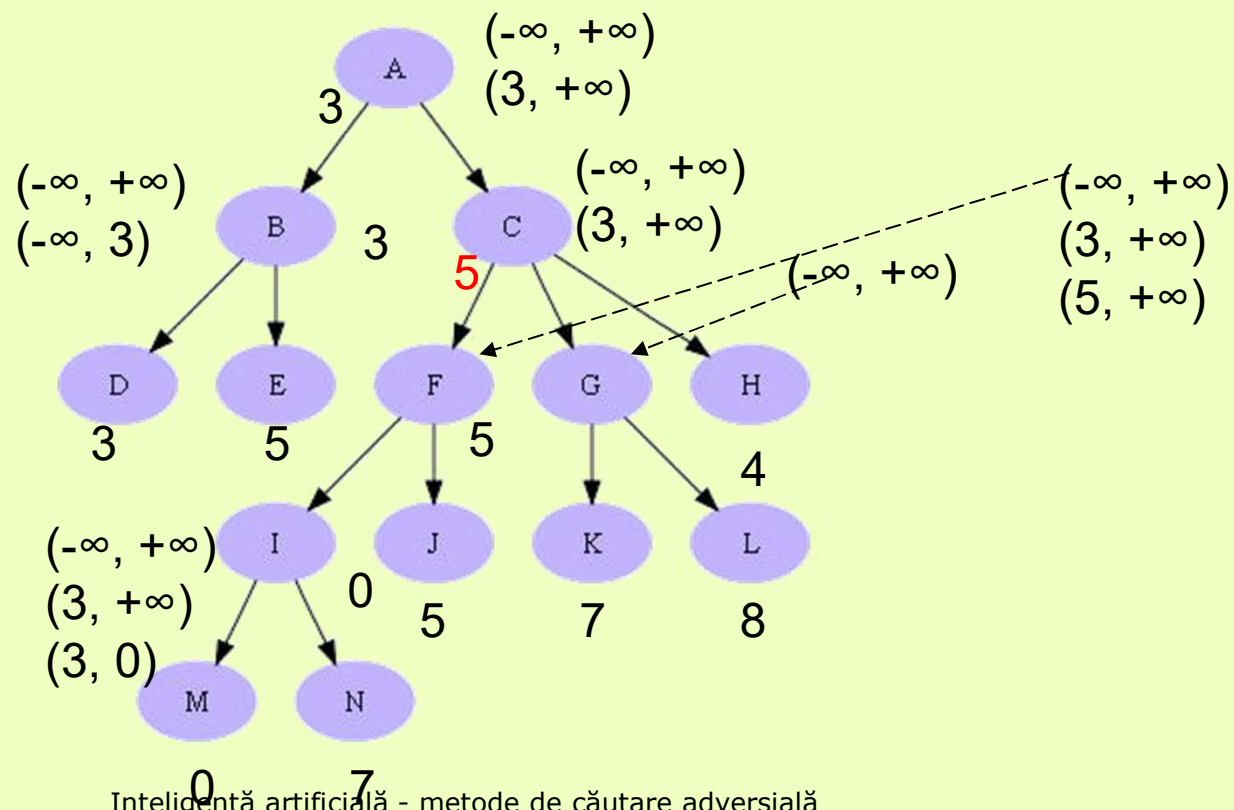
MAX( $\alpha$ )

MIN( $\beta$ )

MAX( $\alpha$ )

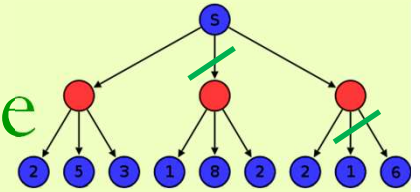
MIN( $\beta$ )

MAX( $\alpha$ )





# Jocurile și căutarea – spațiul de căutare



Strategii de joc → MiniMax cu retezare  $\alpha$ - $\beta$

□ Exemplu

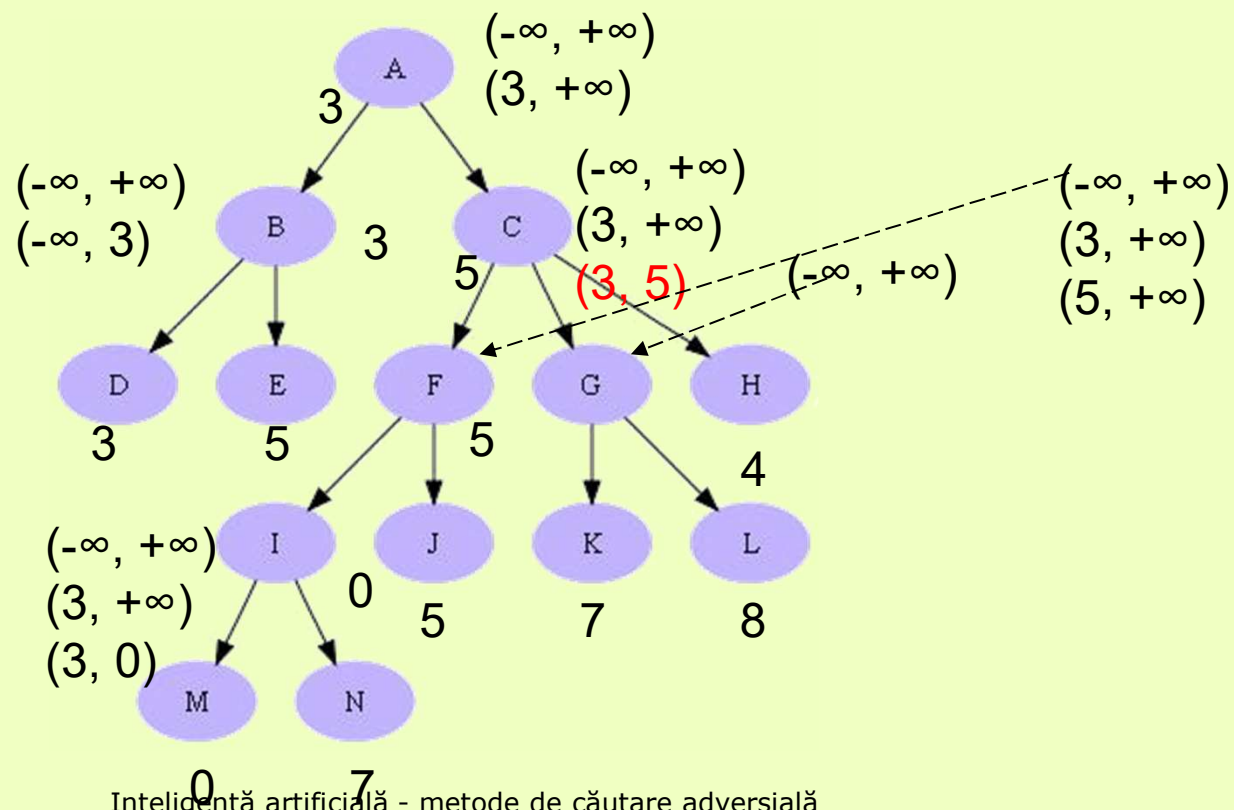
MAX( $\alpha$ )

MIN( $\beta$ )

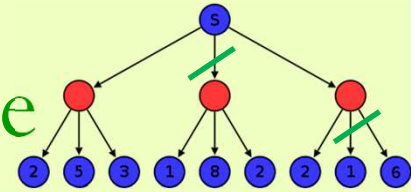
MAX( $\alpha$ )

MIN( $\beta$ )

MAX( $\alpha$ )



# Jocurile și căutarea – spațiul de căutare



Strategii de joc → MiniMax cu retezare  $\alpha$ - $\beta$

□ Exemplu

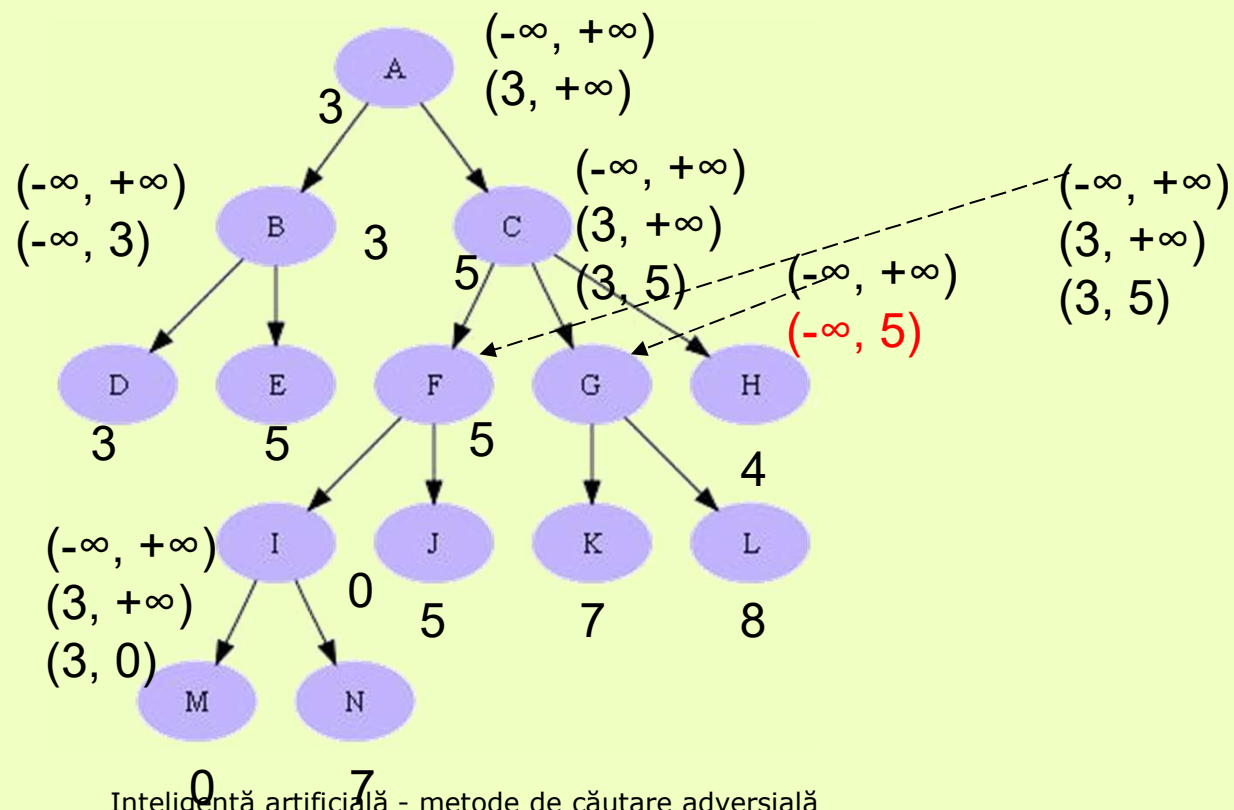
MAX( $\alpha$ )

MIN( $\beta$ )

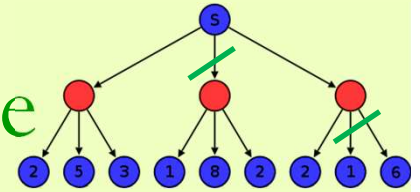
MAX( $\alpha$ )

MIN( $\beta$ )

MAX( $\alpha$ )



# Jocurile și căutarea – spațiul de căutare



Strategii de joc → MiniMax cu retezare  $\alpha$ - $\beta$

□ Exemplu

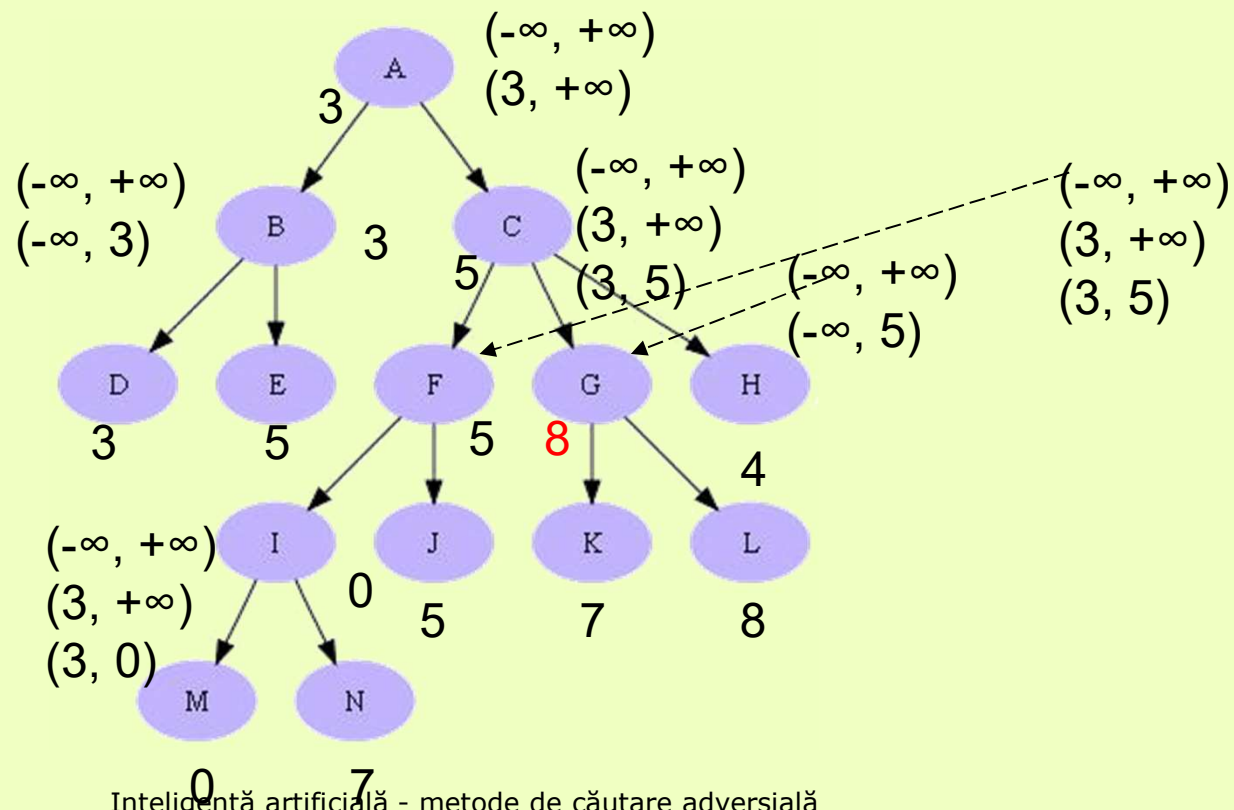
MAX( $\alpha$ )

MIN( $\beta$ )

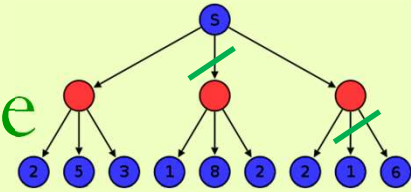
MAX( $\alpha$ )

MIN( $\beta$ )

MAX( $\alpha$ )



# Jocurile și căutarea – spațiul de căutare



Strategii de joc → MiniMax cu retezare  $\alpha$ - $\beta$

□ Exemplu

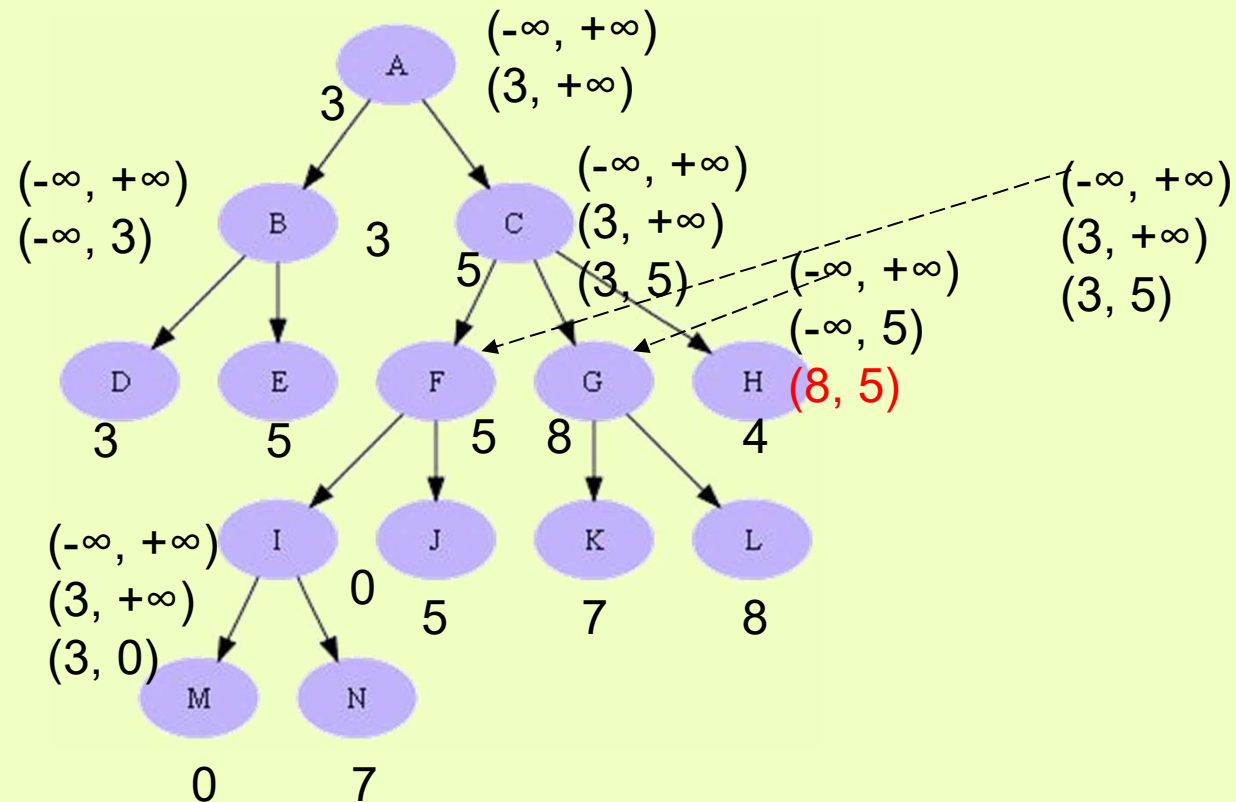
MAX( $\alpha$ )

MIN( $\beta$ )

MAX( $\alpha$ )

MIN( $\beta$ )

MAX( $\alpha$ )



## Exemplu

Diagram illustrating a minimax search tree for a game. The root node is A (Max node). A has children B and C (Min nodes). B has children D and E (Max nodes). C has children F, G, and H (Max nodes). F has children I and J (Min nodes). G has children K and L (Min nodes). I has children M and N (Max nodes). Each node is labeled with its letter and a numerical value. Some nodes are highlighted with boxes containing their minimax values. Dashed arrows indicate the path of the minimax algorithm from the root to the leaf node M.

Node values and minimax values (in boxes):

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

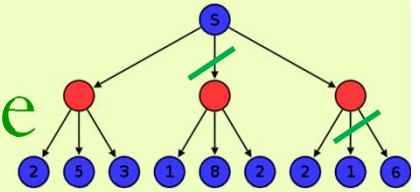
The minimax path is highlighted with dashed arrows: A → C → F → I → M.

Inteligentă artificială - metode de căutare adversală

Inteligentă artificială - metode de căutare adversală

## Exemplu

# Jocurile și căutarea – spațiul de căutare



Strategii de joc → MiniMax cu retezare  $\alpha$ - $\beta$

□ Exemplu

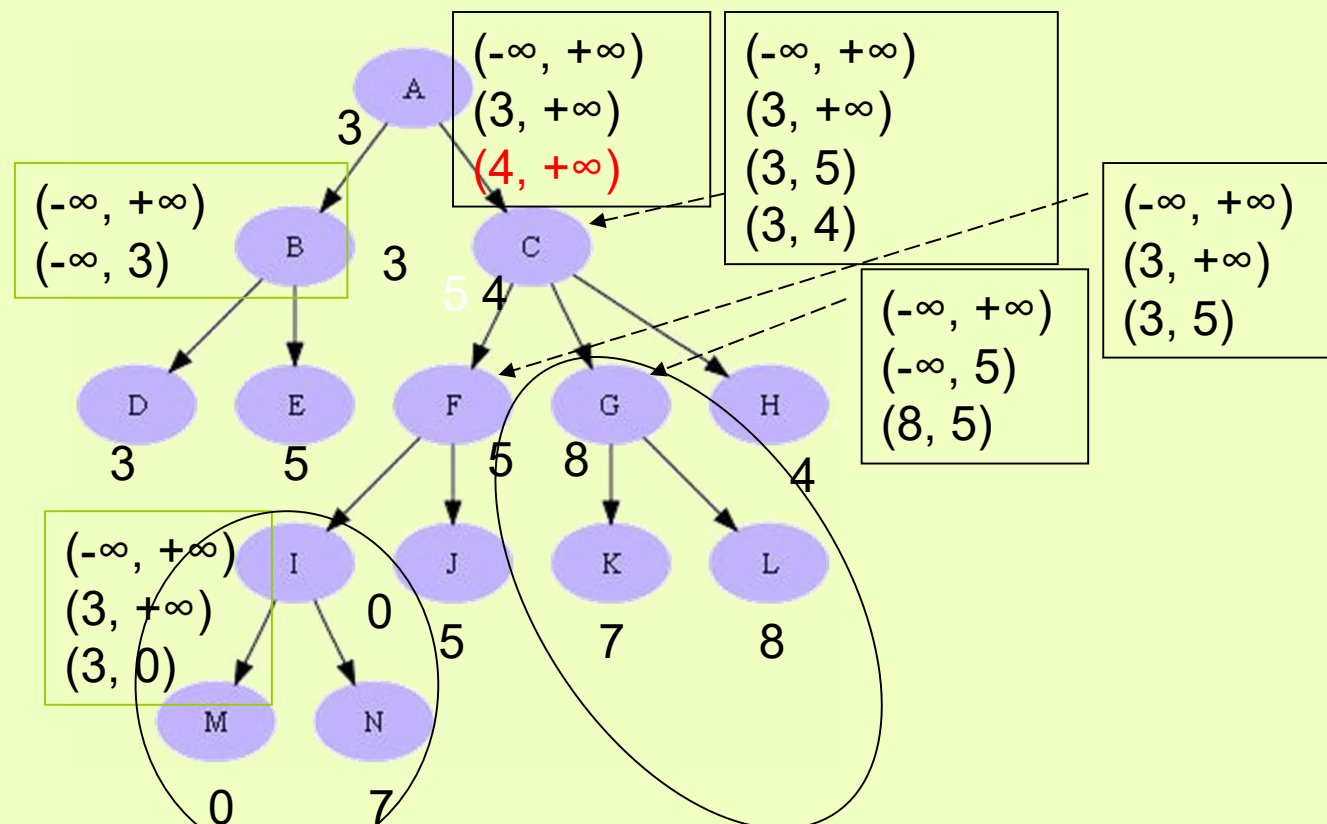
MAX( $\alpha$ )

MIN( $\beta$ )

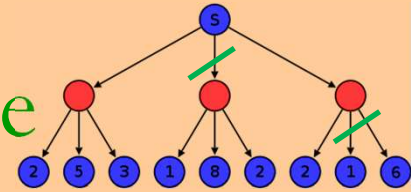
MAX( $\alpha$ )

MIN( $\beta$ )

MAX( $\alpha$ )



# Jocurile și căutarea – spațiul de căutare



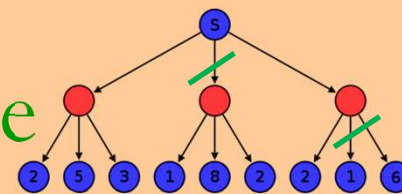
## Strategii de joc → MiniMax cu retezare $\alpha$ - $\beta$

### □ Proprietăți

- Reducerea nu afectează rezultatul final
- O bună ordonare a mutărilor îmbunătățește algoritmul de reducere
- Dacă succesorii sunt puși perfect în ordine (cei mai buni se află primii), atunci complexitatea temporară ar fi  $= O(b^{d/2})$ , în loc de  $O(b^d)$  ca are minimax
- Se poate transforma un program de la nivelul începător la nivelul expert



# Jocurile și căutarea – spațiul de căutare

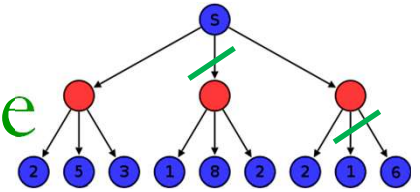


Strategii de joc → MiniMax cu retezare  $\alpha$ - $\beta$

□ MinMax vs. MinMax cu retezare  $\alpha$ - $\beta$

	MinMax	MinMax $\alpha$ - $\beta$
Complexitate temporală	$O(b^m)$	$O(b^{m/2})$ cu ordonare perfectă a nodurilor
Complexitate spațială	$O(b^m)$	$O(2b^{d/2})$ – cel mai bun caz (când unui nod Max îi este generat ca prim copil cel cu valoarea cea mai mare și când lui Min îi este generat ca prim copil cel cu valoarea cea mai mică) $O(b^d)$ – cel mai rău caz (fără retezare)
Completitudine	Da	Da
Optimalitate	Da	Da

# Jocurile și căutarea – spațiul de căutare



- Arbori AndOR, MiniMax, MiniMax cu retezare  $\alpha$ - $\beta$ 
  - Complexitatea mărită → necesitatea unor tehnici eficiente de căutare în rezolvarea jocurilor
    - Jocul de șah
      - $b \sim 35$
      - $d \sim 100$
      - $b^d \sim 35^{100} \sim 10^{154}$  noduri
    - Jocul Tic-Tac-Toe
      - $\sim 5$  mutări legale dintr-un total de 9 mutări
      - $5^9 = 1\,953\,125$
      - $9! = 362\,880$  (dacă computerul mută primul)
      - $8! = 40\,320$  (dacă computerul mută al doilea)
    - Jocul Go
      - $b > 361$  (pentru o tablă de  $19 \times 19$ )

# Jocurile și căutarea – spațiul de căutare



## □ În practică

### ■ Jocul de dame → Chinok

- Chinok îl învinge pe Marion Tinsley în 1994 (după 40 de ani de confruntări)
- Se folosește o bază cu finaluri de joc calculate pentru toate pozițiile unui joc perfect jucat cu cel mult 8 piese (444 bilioane de poziții posibile)

### ■ Jocul de șah → Deep Blue

- Deep Blue l-a învins pe Garry Kasparov în 1997
- Se verifică 200 mil poziții/sec
- Factorul de ramificare se reduce la 6 cu retezarea  $\alpha$ - $\beta$  (față de 35-40)

### ■ Jocul Othello

- Jucătorii umani refuză să joace cu computerele (pt că sunt prea bune) – Moor (1980), Logistello (1997)

### ■ Jocul Go

- Jucătorii umani refuză să joace cu computerele (pt că sunt prea slabe)
- Factorul de ramificare  $> 300 \rightarrow$  necesitatea unor algoritmi bazați pe pattern-uri

### ■ Jocul de table

- BKG (logica fuzzy), TD-Gammon (rețele neuronale artificiale)
- Computerul l-a învins pe campionul lumii, dar pt că a fost norocos

# Jocurile și căutarea – spațiul de căutare



## □ Strategia de joc

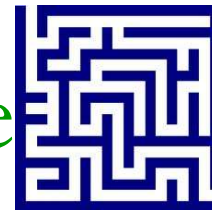
### ■ Pas cu pas

- Ex.: XO, Dame, Șah
- Algoritmi – pot lucra cu structuri:
  - Liniare
    - Strategia simetriei
    - Strategia perechilor
    - Strategia parității
    - Programare dinamică
    - Alte strategii
  - Arborescente
    - Arbori AndOr
    - MiniMax (cu tăieturi Alpha-Beta)

### ■ Completă

- Ex.: ieșirea dintr-un labirint, deplasarea pe o hartă între 2 locații date
- Algoritmi pot să identifice
  - **Un drum optim de la o locație la alta (pathfinding)**
  - O succesiune de acțiuni care să deplaseze jucătorul de la o locație la alta

# Jocurile și căutarea – spațiul de căutare



## Strategii de joc → Pathfinding

### □ Aspecte teoretice

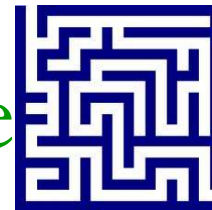
#### ■ Problema

- Identificarea pe o hartă (posibil cu obstacole) a unui drum optim de la o locație la alta
- Unde?
  - pe o hartă
    - harta ca o matrice
    - harta ca un graf (simplu, *mesh*, *quad-tree*)
  - într-un mediu (cunoscut/necunoscut, static/dinamic)
- De către cine?
  - un agent
  - 2 agenți
  - mai mulți agenți

#### ■ Soluția

- Utilizarea unei metode de căutare (optimizare)

# Jocurile și căutarea – spațiul de căutare



## Strategii de joc → Pathfinding

### □ Aspecte teoretice

#### ■ Soluția

##### □ Cum?

- Utilizarea unei metode de căutare (optimizare)

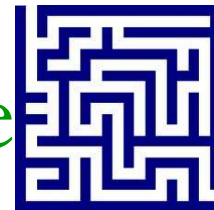
##### □ Care soluție este cea mai bună?

- viteza de calcul,
- lungimea drumului,
- calitatea drumului,
- informația disponibilă

##### □ Dificultăți

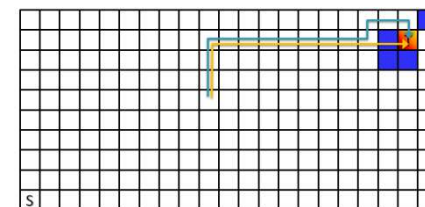
- soluție oferită în timp real
- resurse limitate
- spațiu de căutare imens
- modelarea situațiilor reale implică incertitudini și elemente (teren, unități, agenți) eterogene

# Jocurile și căutarea – spațiul de căutare



Strategii de joc → Pathfinding

- Algoritmi de căutare pentru hărți reprezentate ca “matrici de dale” → labirinturi



- De ce?

- Simplitate

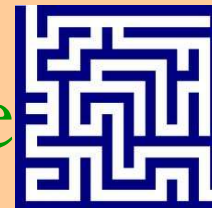
- Caracteristici

- Spațiu de căutare discret
- Dalele sunt pătrate
- Dalelele pot fi traversabile sau blocate
- Mișcări în linie dreaptă de pe o dală pe alta (orizontală, verticală, diagonală)

- Metode

- A\*
  - Algoritmi de tip *Best-first search*
  - Reprezintă un standard în industria jocurilor
- Euristici → Manhattan (gen. Minkowski)
  - Estimarea distanței de la punctul curent la destinație
  - Se ignoră obstacolele

# Jocurile și căutarea – spațiul de căutare



## Strategii de joc → Pathfinding

### □ Algoritmi de căutare pentru hărți reprezentate ca un graf

#### ■ De ce?

- Reprezentarea matriceală → zone mari din hartă care au același cost

#### ■ Caracteristici

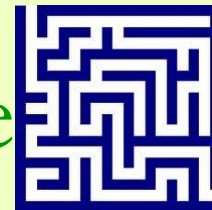
- Spațiu de căutare discret
- "Dalele" pot avea orice formă
- Dalelele pot fi traversabile sau blocate
- Mișcări oarecare de pe o dală pe alta

#### ■ Metode

- Deplasare pe muchii
- Deplasare pe noduri
  - Puncte
  - Marginile poligonului
  - Mijlocul poligonului
- Ex. Dijkstra, Floyd-Warshall, Kruskal, etc



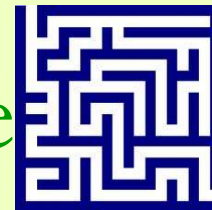
# Jocurile și căutarea – spațiul de căutare



Strategii de joc → Pathfinding

- Îmbunătățiri ale algoritmilor de căutare
  - Euristici mai bune
  - Abstractizări ierarhice
  - Abordări descentralizate

# Jocurile și căutarea – spațiul de căutare



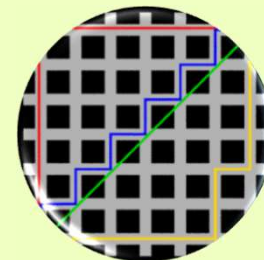
## Strategii de joc → Pathfinding

### □ Îmbunătățiri ale algoritmilor de căutare

#### ■ Euristici mai bune

##### □ Euristici fără memorie

- Ex. Manhattan
- Foarte rapid de calculat
- Nu necesită memorie suplimentară
- Calitate bună uneori



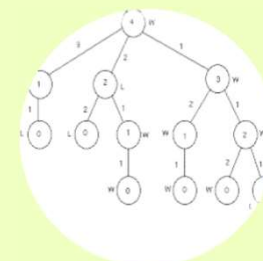
##### □ Euristici bazate pe memorie

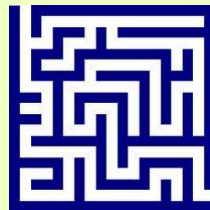
- Ex. euristici bazate pe repere
- Destul de rapide de calculat
- Necesită memorie
- Calitate bună (identificarea drumurilor moarte)



##### □ Informație perfectă

- Foarte costisitoare la calculare și stocare
- Foarte rapidă la utilizat (după ce au fost calculate)
- Ne-practicabile





# Jocurile și căutarea – spațiul de căutare

## Strategii de joc → Pathfinding

### □ Îmbunătățiri ale algoritmilor de căutare

#### ■ Abstractizări ierarhice

##### □ Grafuri în care

- sunt adnotate și fluxurile de mișcare (deplasări în ambele sensuri)
- sunt asociate relații de dominanță între muchii
- Sunt identificate nivele (ex. cameră, casă, oraș, țară)

#### □ Abordări descentralizate

##### □ Ideea de bază:

- Descompunerea problemei în sub-probleme care
  - pot fi rezolvate repede
  - pot fi sub-optimale
  - pot fi incomplete

##### □ Scop

- toate elementele (identificare colaborativă) să ajungă la destinație

##### □ Dificultate

- se mărește spațiul de căutare:  $O(n) \rightarrow O(n^m)$
- factorul de ramificare explodează (de la 4 sau 8 la  $5^m$  sau  $9^m$ )

# Jocurile și căutarea – spațiul de căutare



## Strategii de joc → Planning

### □ Aspecte teoretice

#### ■ Problema

- Planificarea (ordonarea) unor acțiuni
  - mutări, deplasări, alegeri, răsuciri, etc
- Se dau
  - O stare inițială



# Jocurile și căutarea – spațiul de căutare



## Strategii de joc → Planning

### □ Aspecte teoretice

#### ■ Problema

- Planificarea (ordonarea) unor acțiuni
  - mutări, deplasări, alegeri, răsuciri, etc
- Se dau
  - O stare inițială
  - O stare obiectiv (finală)



# Jocurile și căutarea – spațiul de căutare



## Strategii de joc → Planning

### □ Aspecte teoretice

#### ■ Problema

- Planificarea (ordonarea) unor acțiuni
  - mutări, deplasări, alegeri, răsuciri, etc
- Se dau
  - O stare inițială
  - O stare obiectiv (finală)
  - Un set de acțiuni posibile



# Jocurile și căutarea – spațiul de căutare



## Strategii de joc → Planning

### □ Aspecte teoretice

#### ■ Problema

- Planificarea (ordonarea) unor acțiuni
  - mutări, deplasări, alegeri, răsuciri, etc
- Se dau
  - O stare inițială
  - O stare obiectiv (finală)
  - Un set de acțiuni posibile
- Se cere
  - Să se identifice o secvență de acțiuni care transformă starea inițială în starea finală



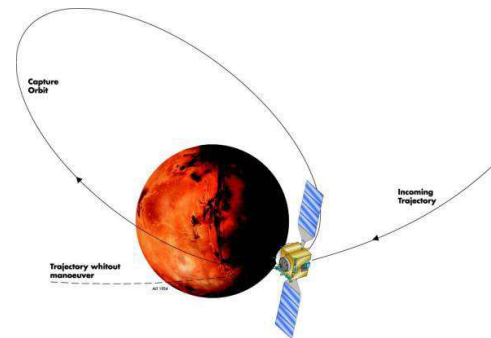
# Jocurile și căutarea – spațiul de căutare



## Strategii de joc → Planning

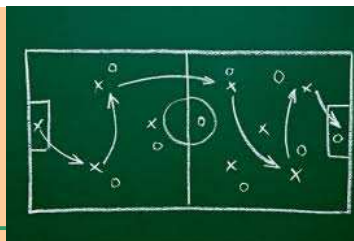
### □ De ce?

- jocuri cu NPC (*First-person shooter, Role-playing, Real-time strategy*)
- Probleme reale de planificare
  - Telescop în spațiu
  - Roboței
  - UAV-uri





# Jocurile și căutarea – spațiul de căutare



## Strategii de joc → Planning

### □ Cum?

#### ■ Algoritmi

- Pentru alegerea unei acțiuni (shooting)
- Pentru evaluarea mediului

#### ■ Abordări

- STRIPS → <http://www.dis.uniroma1.it/~degiacom/didattica/dottorato-stavros-vassos/>
- HTN

#### ■ Simularea comportamentului

- Mașini cu stări finite (FSM)
- Arbori de comportament (Halo 2)
- GOAP (FEAR)

# Recapitulare



- Definirea unui joc
  - Starea inițială (cum sunt așezate inițial elementele în joc)
  - Acțiunile posibile (care sunt mutările permise)
  - Test terminal (care indică terminarea jocului)
  - Funcție utilitate (care spune cine și cât a câștigat)
  
- Strategii de rezolvare a jocurilor
  - Bazate pe explorarea (aproape) completă a arborelui de joc
    - AndOR → cine câștigă
    - MiniMax → cine și cât câștigă
    - MiniMax cu retezare  $\alpha$ - $\beta$  → cine și cât câștigă și ce mutări nu merită să fie efectuate
  - Bazate pe strategii inteligente
    - Strategia simetriei
    - Strategia perechilor
    - Strategia parității
    - Programare dinamică
    - A\* (Pathfinding, Planning)

# Cursul următor

---

## A. Scurtă introducere în Inteligența Artificială (IA)

## B. Rezolvarea problemelor prin căutare

- Definirea problemelor de căutare
- Strategii de căutare
  - Strategii de căutare neinformate
  - Strategii de căutare informate
  - Strategii de căutare locale (Hill Climbing, Simulated Annealing, Tabu Search, Algoritmi evolutivi, PSO, ACO)
  - Strategii de căutare adversială

## C. Sisteme inteligente

- Sisteme care învață singure
  - Arbori de decizie
  - Rețele neuronale artificiale
  - Mașini cu suport vectorial
  - Algoritmi evolutivi
- Sisteme bazate pe reguli
- Sisteme hibride

## Cursul următor – Materiale de citit și legături utile

---

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

---

□ Informațiile prezentate au fost colectate din diferite surse de pe internet, precum și din cursurile de inteligență artificială ținute în anii anteriori de către:

- Conf. Dr. Mihai Oltean –  
[www.cs.ubbcluj.ro/~moltean](http://www.cs.ubbcluj.ro/~moltean)
- Lect. Dr. Crina Groșan -  
[www.cs.ubbcluj.ro/~cgrosan](http://www.cs.ubbcluj.ro/~cgrosan)
- Prof. Dr. Horia F. Pop -  
[www.cs.ubbcluj.ro/~hfpop](http://www.cs.ubbcluj.ro/~hfpop)