Inteligență Artificială

Bogdan Alexe
bogdan.alexe@fmi.unibuc.ro

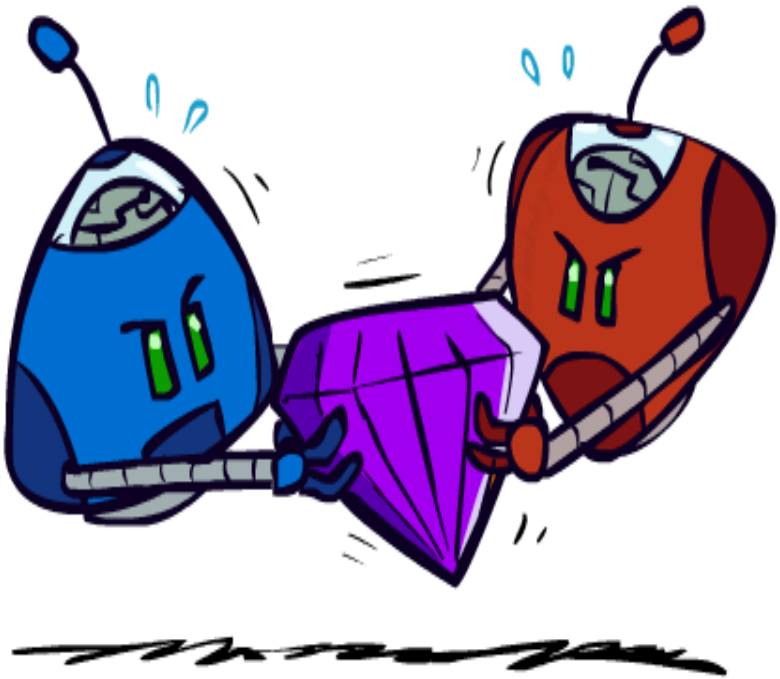
Secția Informatică, anul II, 2022-2023

https://tinyurl.com/Curs-IA-4mai2023

Cuprinsul cursului de azi

1. Jocuri adversariale
2. Algoritmul MINIMAX
3. Algoritmul α-β retezare (α-β prunning)

Jocuri adversariale



Jocuri

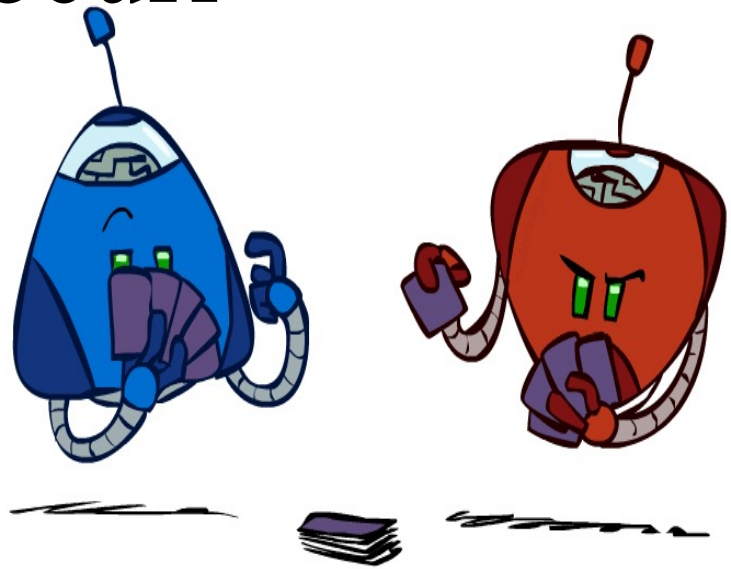
Jocurile reprezintă o arie de aplicaţie pentru algoritmii euristici.

Medii multi-agent:

- fiecare agent trebuie să considere acțiunile celorlalți agenți
- adversar ostil si imprevizibil, incontrolabil care introduce incertitudine
- medii competitive (joc = căutare adversarială)

Probleme de căutare adversarială sunt diferite de probleme de căutare, unde soluția este o secvență de acțiuni care garantează obținerea scopului.

Tipuri de jocuri



- multe tipuri de jocuri!

- dimensiuni:
  - deterministic sau stochastic? (dacă următoarea stare a jocului este în întregime determinată de starea curentă și de acțiunile agentului atunci avem un joc determinist)
  - unul, doi, sau mai mulți jucători?
  - joc de sumă zero?
  - complet sau parțial observabil? (se cunoaște întreaga stare)

- vrem să găsim algoritmi pentru a calcula o strategie ce recomandă o mutare din fiecare stare

Jocul de dame

Complet observabil vs parțial observabil
Determinist vs stochastic

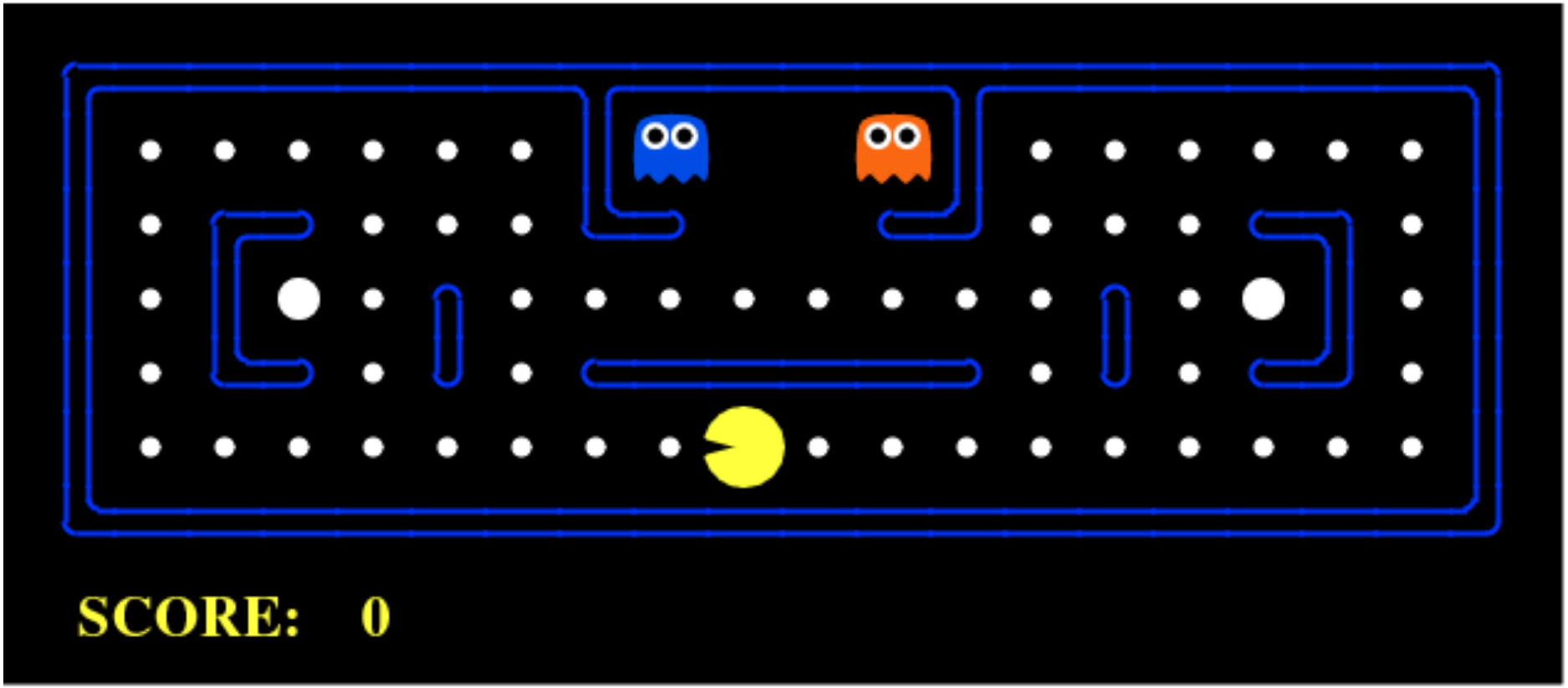
Complet observabil
Determinist

Jocul de poker

Complet observabil vs parțial observabil
Determinist vs stochastic

Parțial observabil
Stochastic

PAC MAN



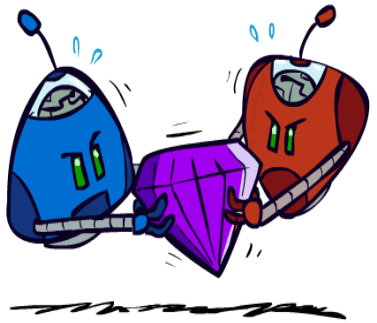
Complet observabil vs parțial observabil
Determinist vs stochastic

Complet observabil
Stochastic

Tipuri de jocuri

	determinist	stochastic
Complet observabil	sah, dame, go	table, monopoly
Parțial observabil	avioane	poker, catan

- dimensiuni:
  - deterministic sau stochastic?
  - unul, doi, sau mai mulți jucători?
  - joc de sumă zero?
  - complet sau parțial observabil? (se cunoaște întreaga stare)

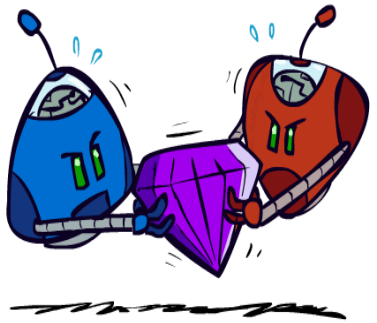


- vrem să găsim algoritmi pentru a calcula o strategie ce recomandă o mutare din fiecare stare

Tipuri de jocuri

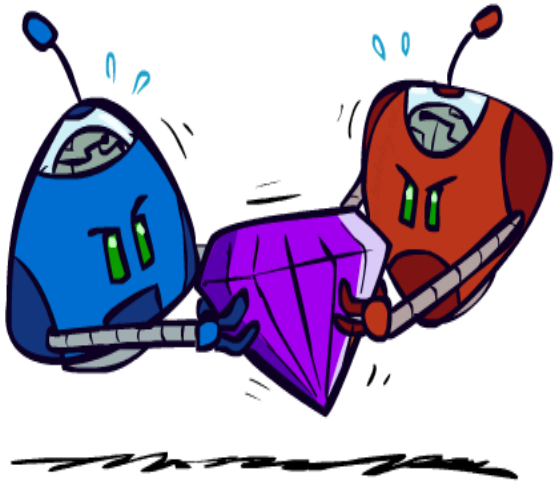
	determinist	stochastic
Complet observabil	sah, dame, go	table, monopoly
Parțial observabil	avioane	poker, catan

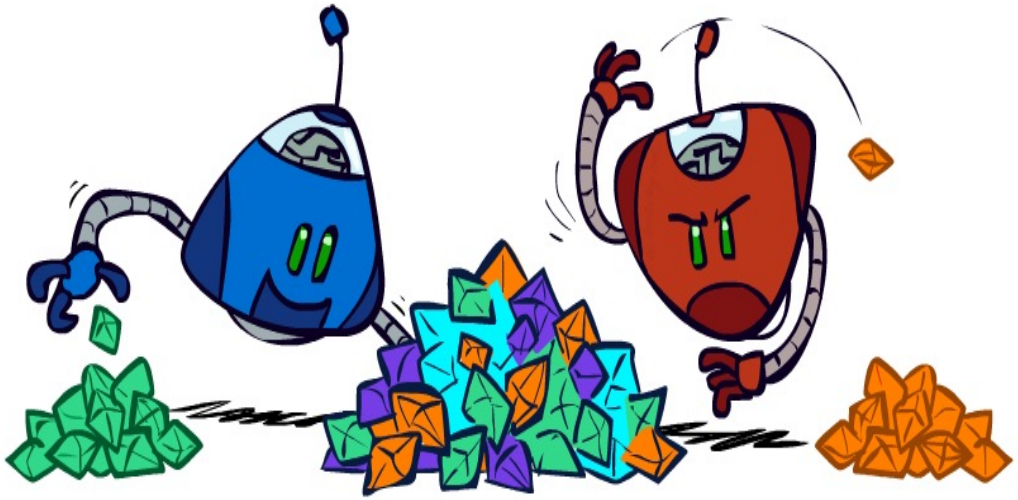
- dimensiuni:
  - deterministic sau stochastic?
  - unul, doi, sau mai mulți jucători?
  - joc de sumă zero?
  - complet sau parțial observabil? (se cunoaște întreaga stare)



- vrem să găsim algoritmi pentru a calcula o strategie ce recomandă o mutare din fiecare stare

Jocuri cu sumă zero





- Jocuri cu sumă zero
  - agenții au utilități (evaluari ale rezultatelor) opuse
  - un agent dorește maximizarea utilității (agentul MAX), celălalt minimizarea (agentul MIN)
  - mediu adversarial pur competitiv
- Jocuri generale
  - agenții au utilități independente
  - cooperare, indiferență, competiție

O definire formală a jocurilor

Tipul de jocuri la care ne vom referi în continuare este acela al jocurilor de două persoane cu informaţie perfectă sau completă. În astfel de jocuri:

- **există doi jucători care efectuează mutări în mod alternativ;**
- **ambii jucători dispun de informaţia completă asupra situaţiei curente a jocului;**
- **jocul se încheie atunci când este atinsă o poziţie calificată ca fiind "terminală" de către regulile jocului - spre exemplu, "mat" în jocul de şah.**

Un asemenea joc poate fi reprezentat printr-un *arbore de joc* în care nodurile corespund situaţiilor (stărilor), iar arcele corespund mutărilor. Situaţia iniţială a jocului este reprezentată de nodul rădăcină, iar frunzele arborelui corespund poziţiilor terminale.

Un joc poate fi definit, în mod formal, ca fiind *un anumit tip de problemă de căutare având următoarele componente*:

- ***starea iniţială*, care include poziţia de pe tabla de joc şi o indicaţie referitoare la cine face prima mutare;**
- ***o mulţime de operatori*, care definesc mişcările permise ("legale") unui jucător;**
- ***un test terminal*, care determină momentul în care jocul ia sfârşit;**
- ***o funcţie de utilitate* (numită şi *funcţie de plată*), care acordă o valoare numerică rezultatului unui joc; în cazul jocului de şah, spre exemplu, rezultatul poate fi *câştig*, *pierdere* sau *remiză*, situaţii care pot fi reprezentate prin valorile 1, -1 sau 0.**

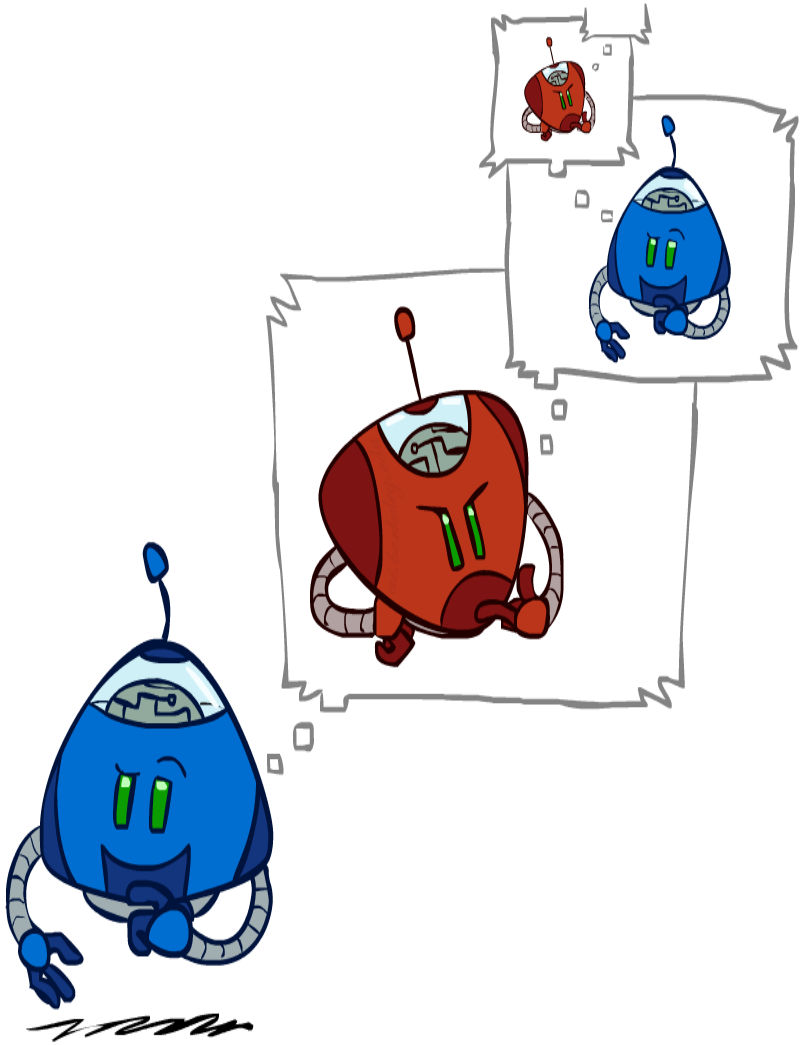
- **Vom lua în consideraţie cazul general al unui joc cu doi jucători, pe care îi vom numi MAX şi respectiv MIN.**

- **MAX va face prima mutare, după care jucătorii vor efectua mutări pe rând, până când jocul ia sfârşit.**

- **La finalul jocului vor fi acordate puncte jucătorului câştigător (sau vor fi acordate anumite penalizări celui care a pierdut).**

- **Dacă un joc ar reprezenta o problemă standard de căutare, atunci acţiunea jucătorului MAX ar consta din căutarea unei secvenţe de mutări care conduc la o stare terminală reprezentând o stare câştigătoare (conform funcţiei de utilitate) şi din efectuarea primei mutări aparţinând acestei secvenţe.**
- **Acţiunea lui MAX interacţionează însă cu cea a jucătorului MIN. Prin urmare, MAX trebuie să găsească *o strategie* care va conduce la o stare terminală câştigătoare, indiferent de acţiunea lui MIN. Această strategie include mutarea corectă a lui MAX corespunzătoare fiecărei mutări posibile a lui MIN.**
- **În cele ce urmează, vom arăta cum poate fi găsită strategia optimă (sau raţională), deşi în realitate, cel mai probabil, nu vom dispune de timpul necesar pentru a o calcula.**

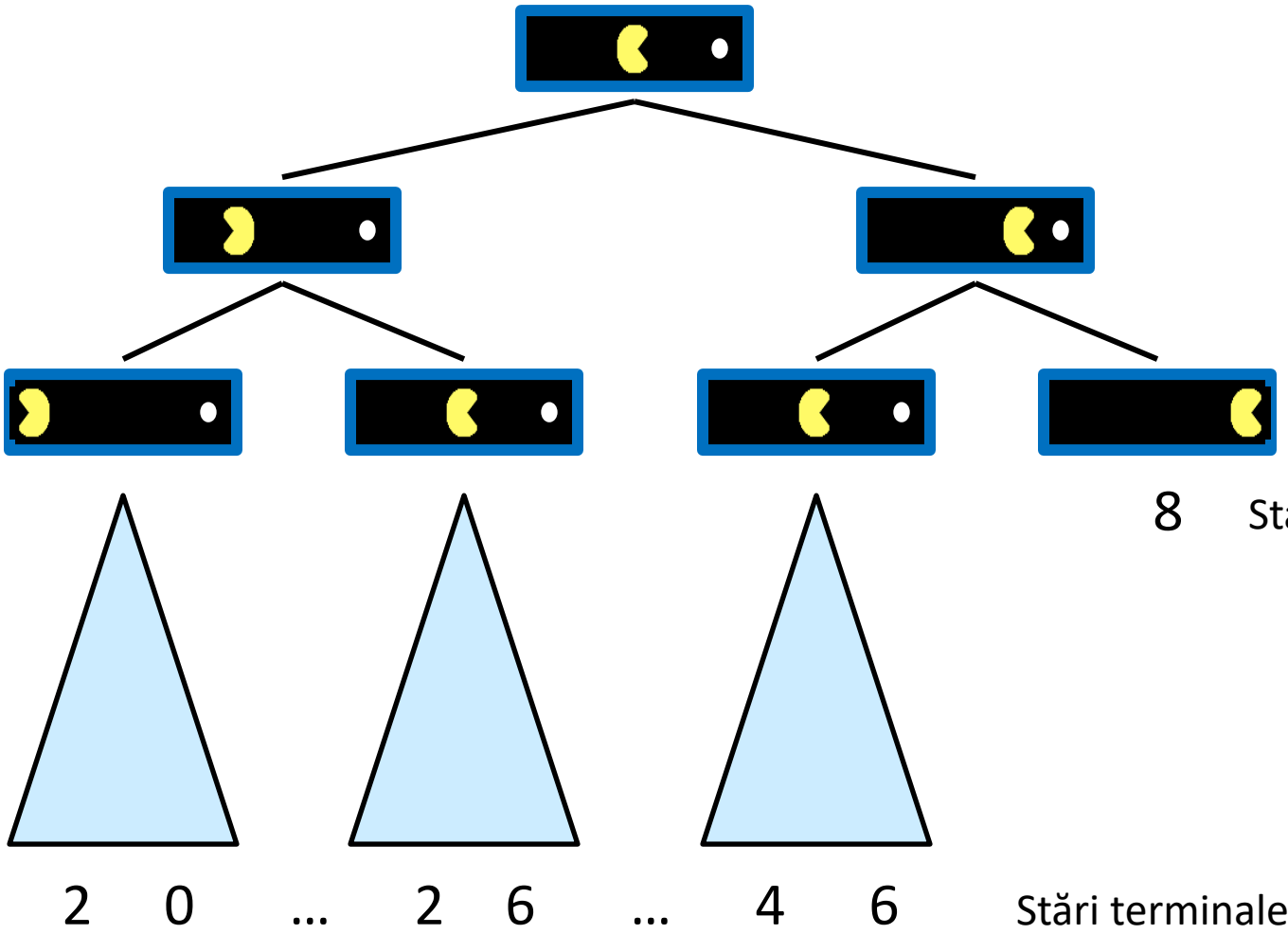
Găsirea unei strategii optime



Arbore de căutare într-un mediu cu un singur agent

Fiecare mutare: -1 punct

Mânânc punctul alb: + 10 puncte

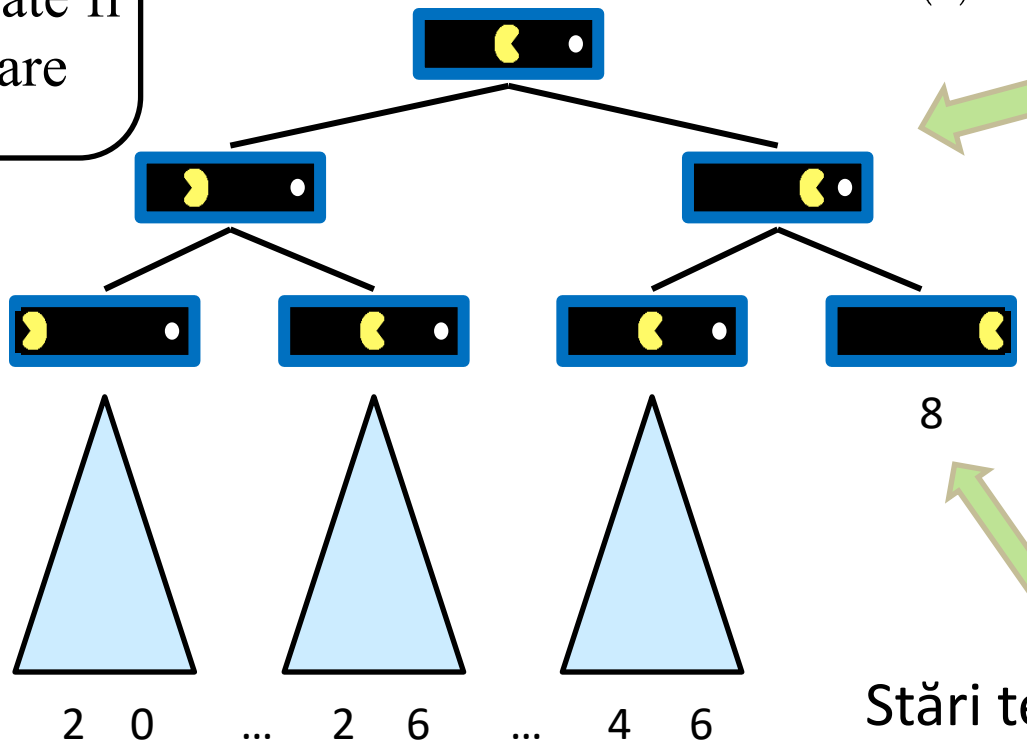


Valoarea unei stări

Valoarea unei stări: cea mai bună utilitate (rezultat) care poate fi atinsă din acea stare

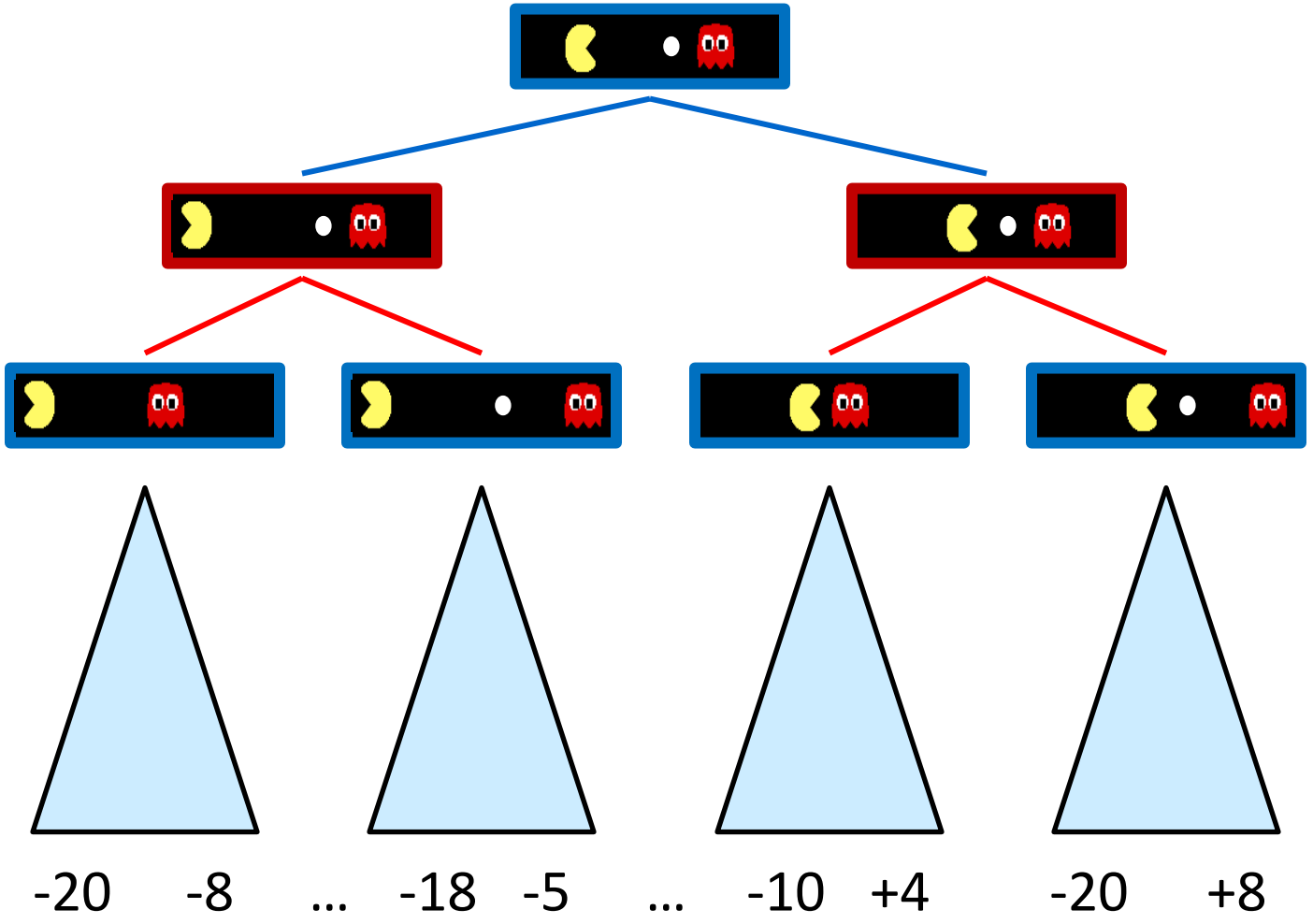
Stări neterminale:

$$V(s) = \max_{s' \in \text{children}(s)} V(s')$$



Stări terminale:
V(s) = cunoscută, dată de regulile jocului

Arbore de joc în medii cu doi agenți, deterministic, mutări pe rând



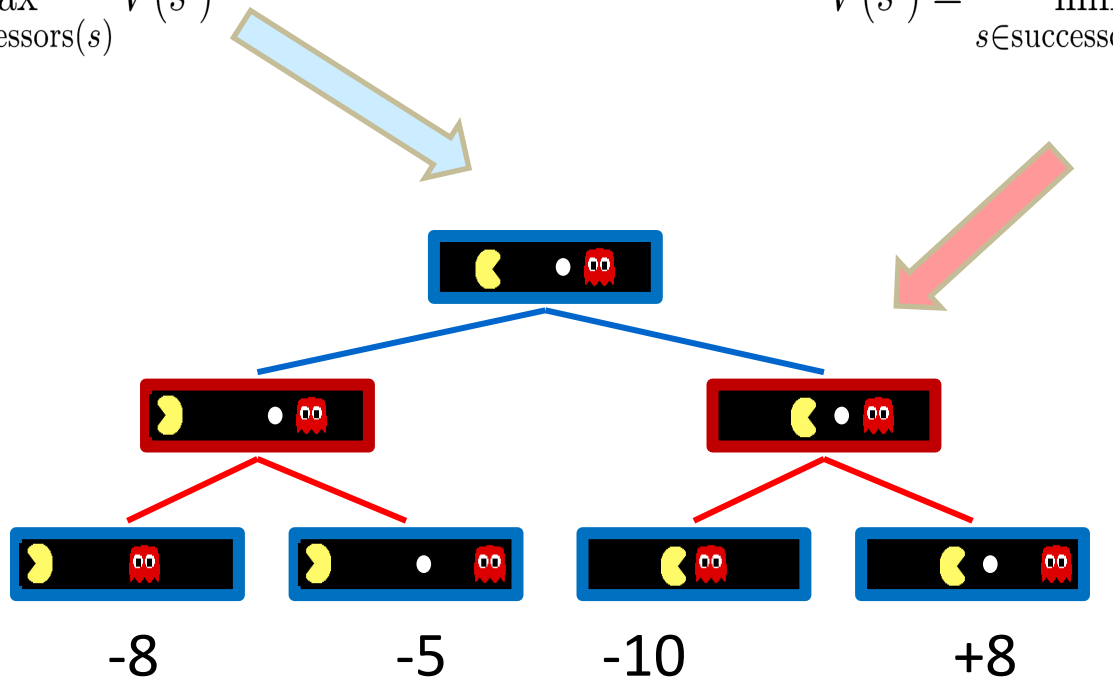
Valoare MiniMax a unei stări

Stări controlate de MAX

$$V(s) = \max_{s' \in \text{successors}(s)} V(s')$$

Stări controlate de MIN:

$$V(s') = \min_{s \in \text{successors}(s')} V(s)$$

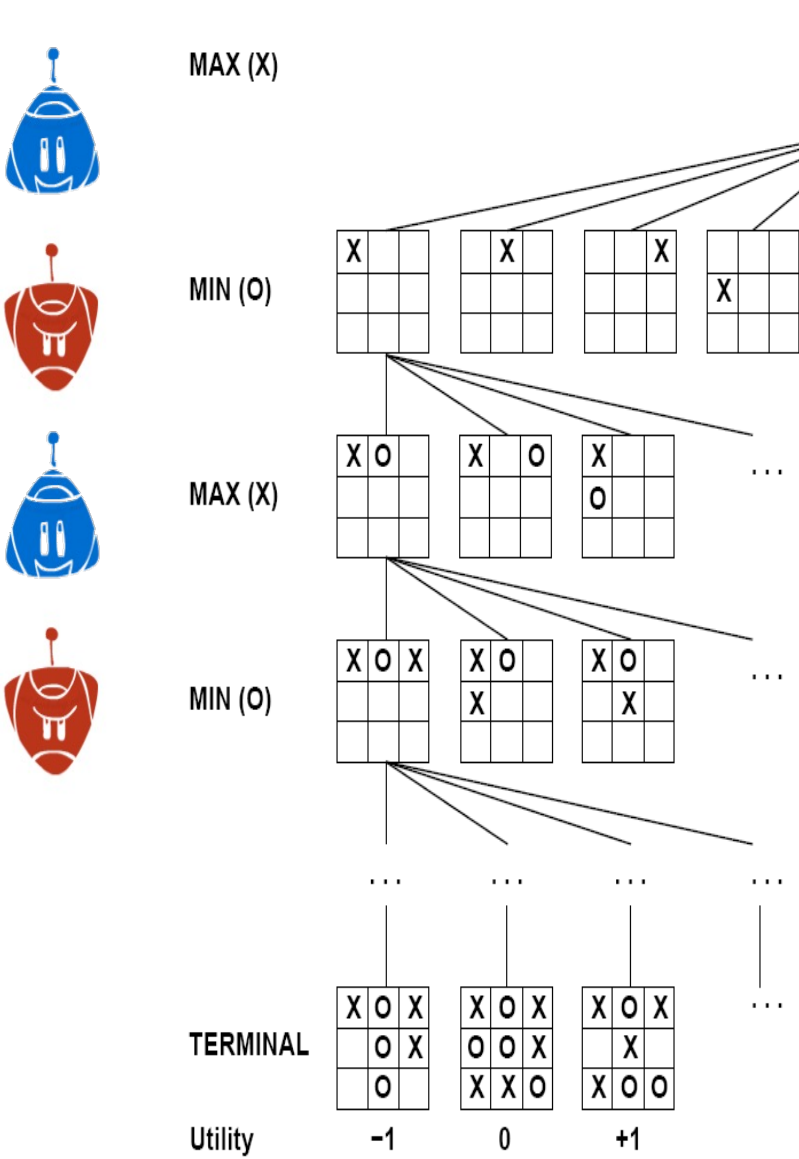


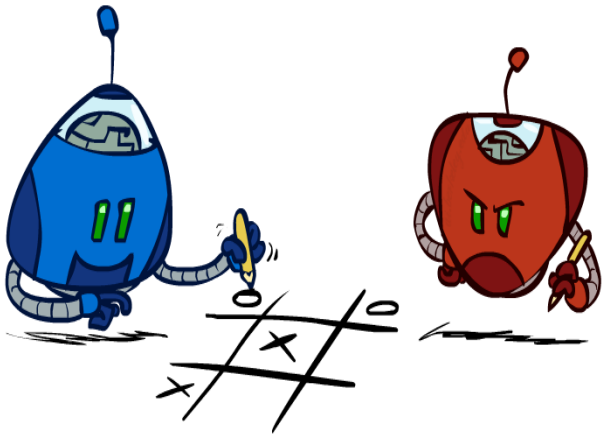
Stări terminale: V(s) = cunoscută, dată de regulile jocului

Arbore de joc

- nodurile corespund situațiilor (stărilor)
- situația inițială a jocului dată de nodul rădăcină
- arcele corespund mutărilor
- frunzele corespund situaților terminale (s-a terminat jocul)
- 2 jucători: MAX (de obicei este cel care face prima mutare) și MIN

Arbore de joc pentru X și 0





Cuprinsul cursului de azi

1. Jocuri adversariale

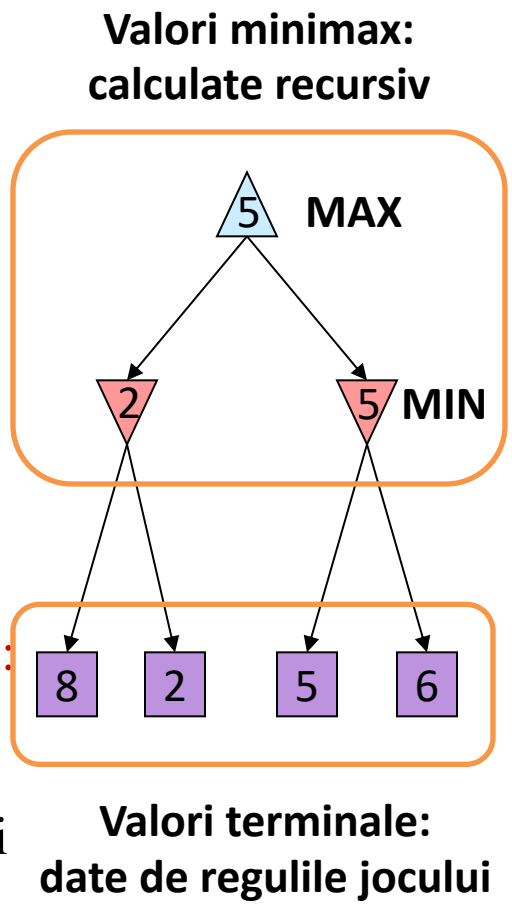
2. **Algoritmul MINIMAX**

3. Algoritmul α-β retezare (α-β prunning)

Algoritmul MiniMax pentru căutare adversarială

- Jocuri de sumă zero cu doi jucători, deterministe, cu informație perfectă
  - X și 0, șah, dame
  - un jucător (MAX) maximizează utilitatea
  - celălalt jucător (MIN) minimizează utilitatea

- Căutare MiniMax:
  - arbore de căutare în spațiul stărilor
  - jucătorii mută pe rând
  - calculăm pentru fiecare nod valoarea minimax: cea mai bună utilitate care poate fi obținută împotriva unui adversar care joacă perfect
  - determină strategia optimă corespunzătoare lui MAX, care este cea mai bună primă mutare



Algoritmul MiniMax - descriere

1. Generează întreg arborele de joc până la stările terminale

2. Aplică funcția de utilitate fiecărei stării terminale - obține valoarea stării

3. Deplasează-te înapoi în arbore, de la nodurile-frunze spre nodul-rădăcină, determinând, corespunzător fiecărui nivel al arborelui, valorile care reprezintă utilitatea nodurilor aflate la acel nivel. Propagarea acestor valori la niveluri anterioare se face prin intermediul nodurilor-părinte succesive, conform următoarei reguli:

   - dacă starea-părinte este un nod de tip MAX, atribuie-i maximul dintre valorile avute de fii săi;

   - dacă starea-părinte este un nod de tip MIN, atribuie-i minimul dintre valorile avute de fii săi;

4. Ajuns în nodul-rădăcină, alege pentru MAX acea mutare care conduce la valoarea maximă. Mutarea se numește ***decizia minimax*** - maximizează utilitatea, în ipoteza că oponentul joacă perfect cu scopul de a o minimiza.

Algoritmul MiniMax - formalizare

- jucătorul care trebuie să mute într-o stare: PLAYER(s)
- acțiuni (mutări) legale într-o stare: ACTIONS(s)
- modelul de tranziție (funcția succesor): RESULT(s,a)
- test terminal : TERMINAL-TEST(s)
- o funcție de utilitate – asociază o valoare numerică stărilor terminale: UTILITY(s)

$$\textsc{Minimax}(s) = \begin{cases} \textsc{Utility}(s) & \text{if } \textsc{Terminal-Test}(s) \\ \max_{a \in Actions(s)} \textsc{Minimax}(\textsc{Result}(s,a)) & \text{if } \textsc{Player}(s) = \textsc{max} \\ \min_{a \in Actions(s)} \textsc{Minimax}(\textsc{Result}(s,a)) & \text{if } \textsc{Player}(s) = \textsc{min} \end{cases}$$

Implementare MiniMax

```
def value(stare):
    dacă stare este o stare terminală: returnează utilitatea stării
    dacă următorul agent care mută este MAX: returnează max-value(stare)
    dacă următorul agent care mută este MIN: returnează min-value(stare)
```

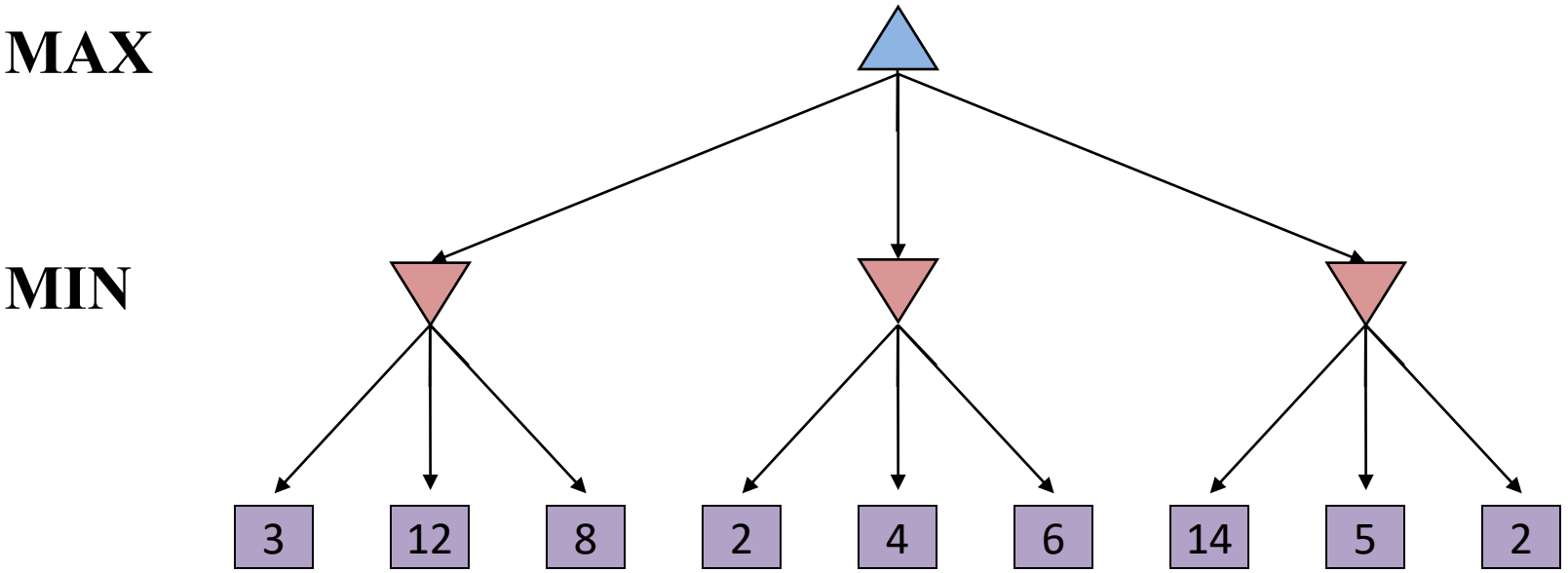
```
def max-value(stare):
    initializează v = -∞
    pentru fiecare successor al starii:
        v = max(v, value(successor))
    returnează v
```

```
def min-value(state):
    initializează v = +∞
    pentru fiecare succesor al stării:
        v = min(v, value(successor))
    returnează v
```

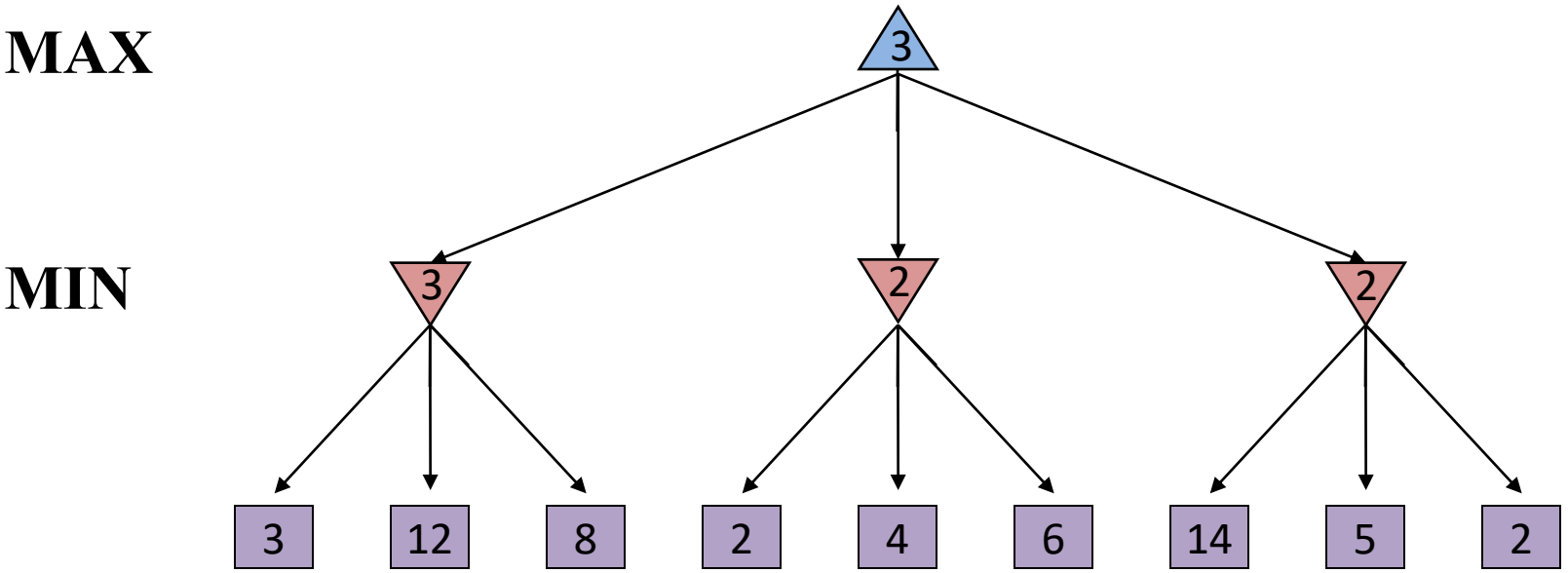
$$V(s) = \max_{s' \in \text{successors}(s)} V(s')$$

$$V(s') = \min_{s \in \text{successors}(s')} V(s)$$

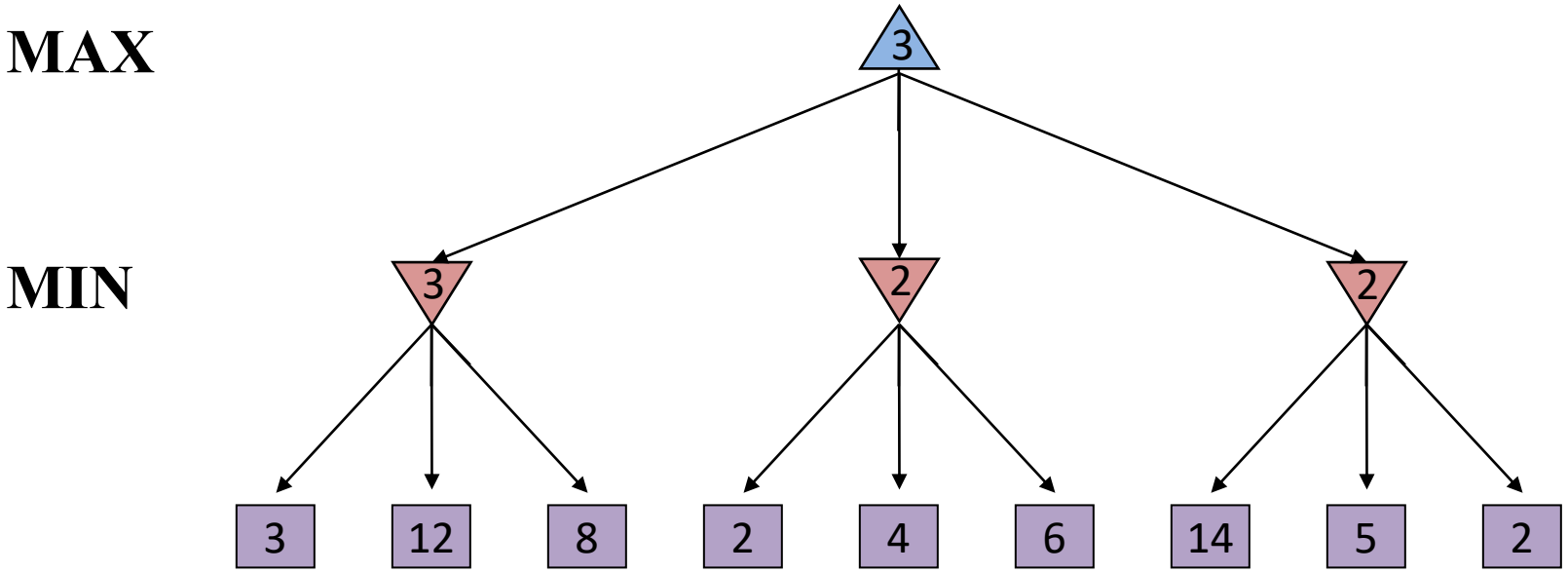
Exemplul MiniMax



Exemplul MiniMax



Exemplul MiniMax

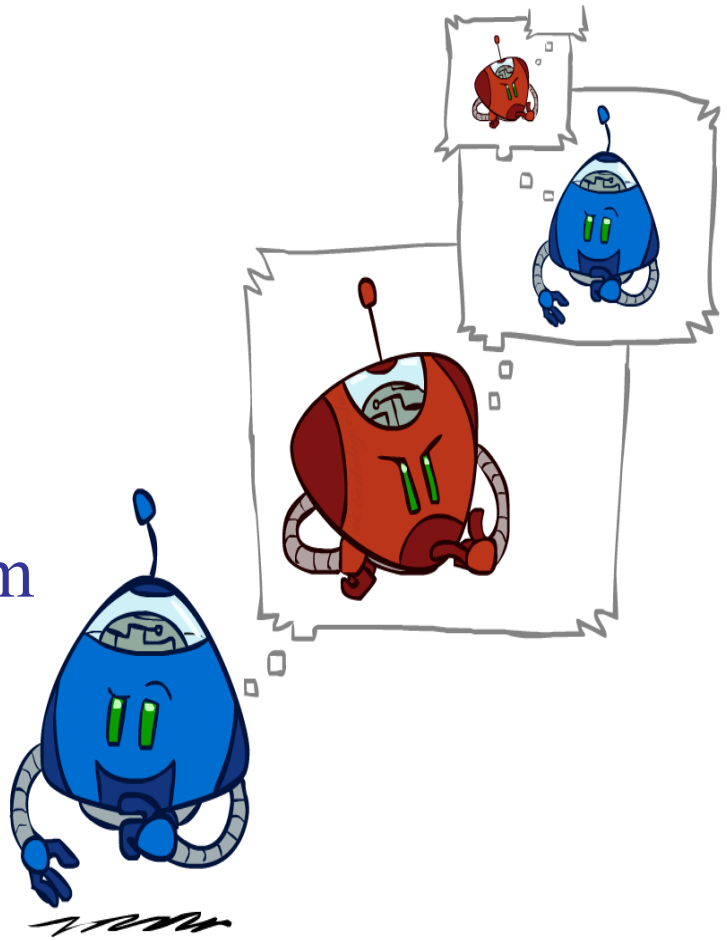


- valorile pozițiilor de la ultimul nivel sunt determinate de către funcția de utilitate și se numesc valori statice (se realizează evaluări statice).
- valorile minimax ale nodurilor interne sunt calculate în mod dinamic, în manieră bottom-up, nivel cu nivel, până când este atins nodul-rădăcină.
- valoarea rezultată este 3 și prin urmare cea mai bună mutare a lui MAX din poziția curentă este mutarea la stânga. Cel mai bun răspuns al lui MIN este mutarea la stânga. Această secvență a jocului poartă denumirea de variație principală. Ea definește jocul optim de tip minimax pentru ambele părți.
- se observă ca valoarea pozițiilor de-a lungul variației principale nu variază. Prin urmare, ***mutările corecte sunt cele care conservă valoarea jocului.***

Proprietăți ale algoritmului MiniMax

- Cât de eficient este MiniMax?
  - Căutare exhaustivă în manieră depth-first
  - timp: $O(b^m)$
  - spațiu: O(bm)

- Exemple: pentru șah, $b \approx 35$, $m \approx 100$
  - o soluție exactă este imposibilă
  - oare chiar trebuie să explorăm întreg arborele?



Limitări date de resurse ale algoritmului MiniMax

1. **Generează întreg arborele de joc până la stările terminale**

2. Aplică funcția de utilitate fiecărei stării terminale - obține valoarea stării

3. Deplasează-te înapoi în arbore, de la nodurile-frunze spre nodul-rădăcină, determinând, corespunzător fiecărui nivel al arborelui, valorile care reprezintă utilitatea nodurilor aflate la acel nivel. Propagarea acestor valori la niveluri anterioare se face prin intermediul nodurilor-părinte succesive, conform următoarei reguli:

   - dacă starea-părinte este un nod de tip MAX, atribuie-i maximul dintre valorile avute de fii săi;

   - dacă starea-părinte este un nod de tip MIN, atribuie-i minimul dintre valorile avute de fii săi;

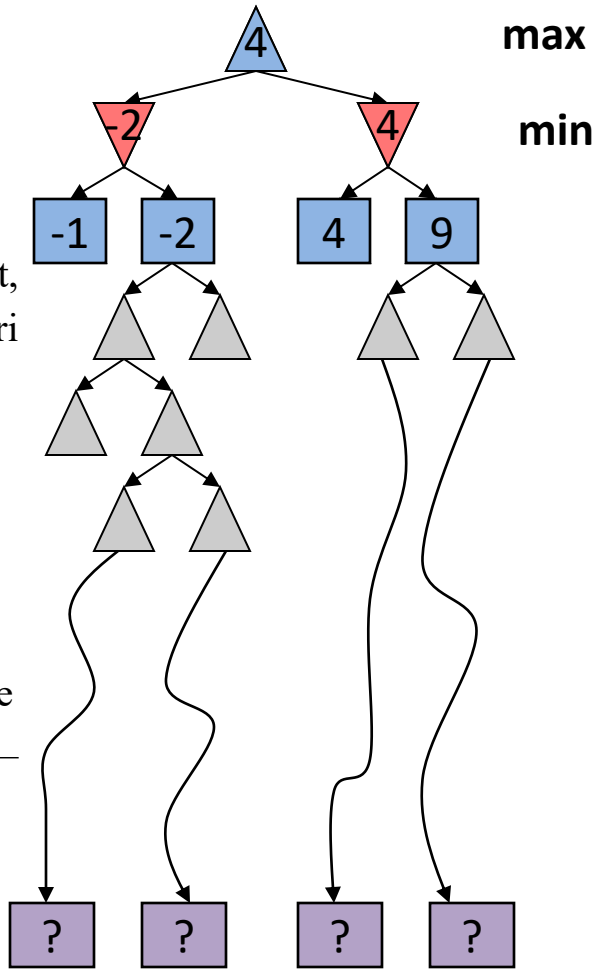
4. Ajuns în nodul-rădăcină, alege pentru MAX acea mutare care conduce la valoarea maximă. Mutarea se numește ***decizia minimax*** - maximizează utilitatea, în ipoteza că oponentul joacă perfect cu scopul de a o minimiza.

Limitări date de resurse

- problemă: în jocuri reale, nu putem genera tot arborele până la frunze din cauza resurselor limitate (timp + spațiu)

- soluție posibilă: căutare în adâncime limitată
  - caută până la o anumită adâncime în arborele de joc
  - înlocuiește utilitățile stărilor terminale (care sunt calculate exact, după regulile jocului) cu o funcție de evaluare a utilității unor stări neterminale (această funcție aproximează utilitatea)

- exemplu:
  - avem la dispoziție 100 de secunde petru a realiza mutarea;
  - putem explora 10000 noduri / secundă;
  - în 100 de secunde putem explora 1 Milion de noduri pentru mutare
  - algoritmul α-β retezare (îl prezentăm azi) ajunge la adâncime 8 – performanță bună în șah (stockfish);



- garanția de optimalitate dispare;
- cu cât pot analiza mai multe noduri cu atât mutarea este mai bună;
- uneori pot folosi și un algoritm de căutare iterativea/incrementală în adâncime

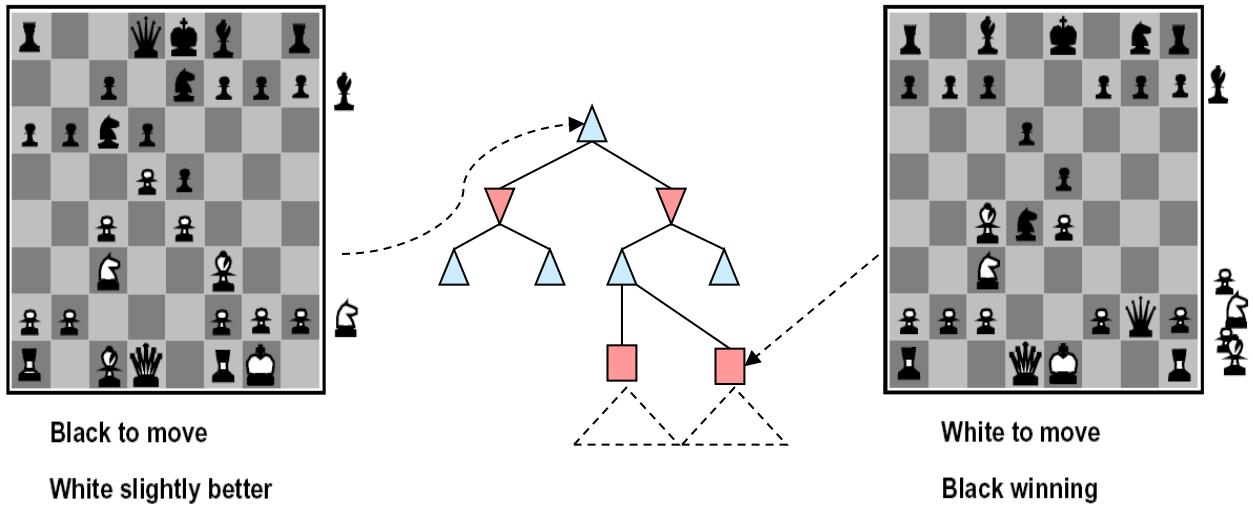
Influența adâncimii pentru funcția de evaluare

- Funcțiile de evaluare sunt imperfecte

- Cu cât merg mai adâncime în arbore și apelez funcția de evaluare a unei stări neterminale mai târziu cu atât am șanse să greșesc mai puțin.

Funcții de evaluare - șah

- Funcțiile de evaluare asociază un scor stărilor neterminale (este folosită de căutarea în adâncime limitată, căutarea în adâncime iterativă)



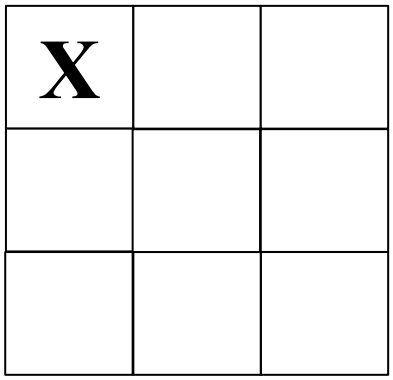
- o funcție de evaluare ideală returnează valoarea MiniMax a stării.
- diverse funcții de evaluare specifice pentu fiecare joc: pentru șah se consideră funcții liniare în care ponderăm piesele de pe tablă

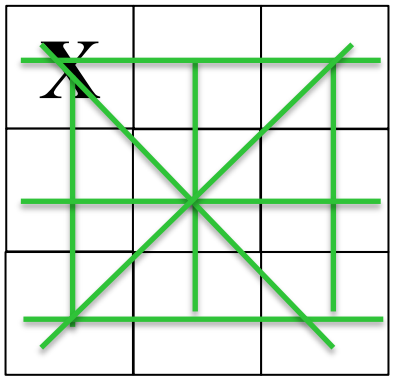
$$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$

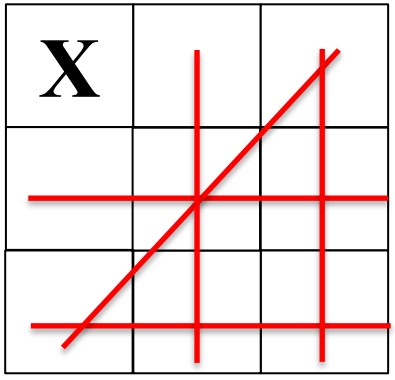
- unde, $f_1(s)$ = (#regine_albe – #regine_negre), $w_1 = 100$, etc.

Funcții de evaluare – X și 0

- o funcție de evaluare ideală returnează valoarea MiniMax a stării.
- diverse funcții de evaluare specifice pentru fiecare joc:



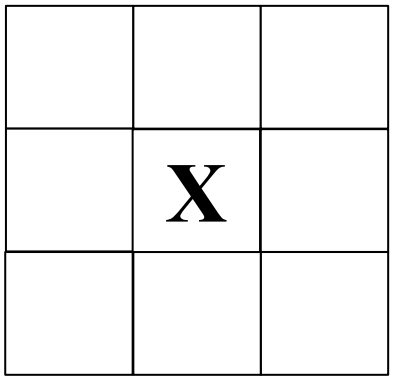


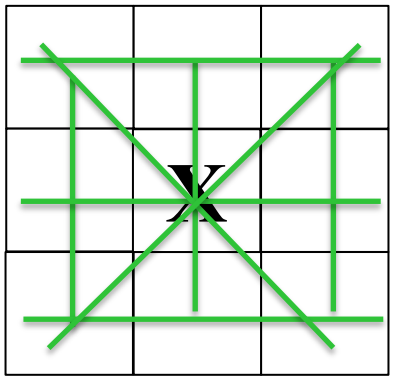


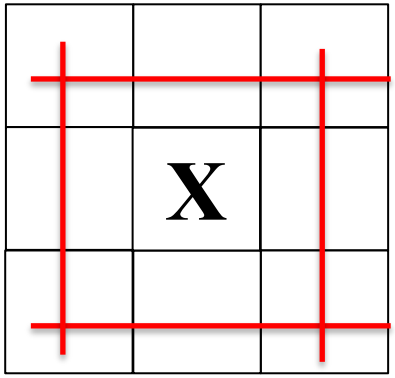
- exemplu de funcție de evaluare pentru X și 0: diferența dintre numărul de soluții disponibile pentru MAX și numărul de soluții disponibile pentru MIN
- în exemplul de mai sus: 8-5 = 3

Funcții de evaluare – X și 0

- o funcție de evaluare ideală returnează valoarea MiniMax a stării.
- diverse funcții de evaluare specifice pentru fiecare joc:



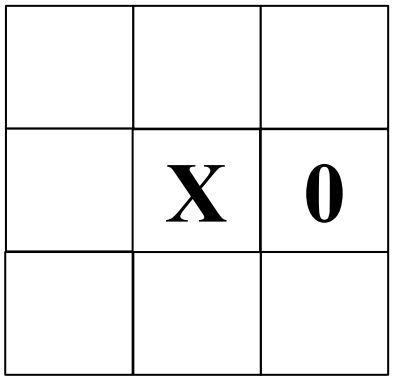


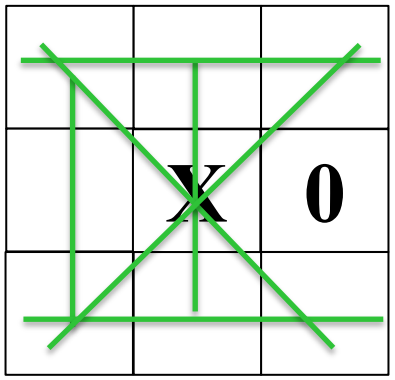


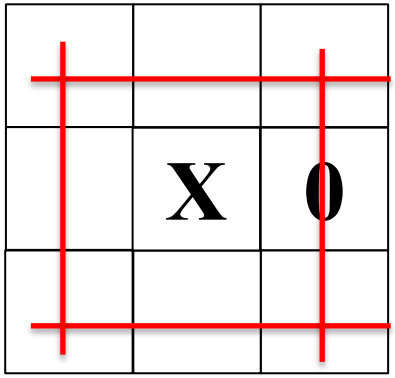
- exemplu de funcție de evaluare pentru X și 0: diferența dintre numărul de soluții disponibile pentru MAX și numărul de soluții disponibile pentru MIN
- În exemplul de mai sus: 8-4 = 4

Funcții de evaluare – X și 0

- o funcție de evaluare ideală returnează valoarea MiniMax a stării.
- diverse funcții de evaluare specifice pentru fiecare joc:



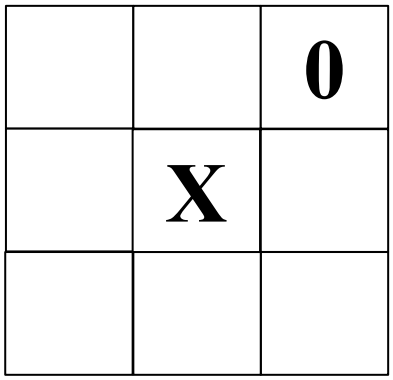


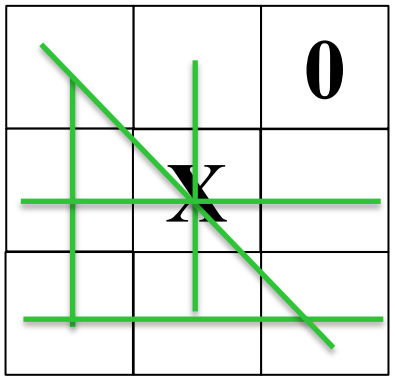


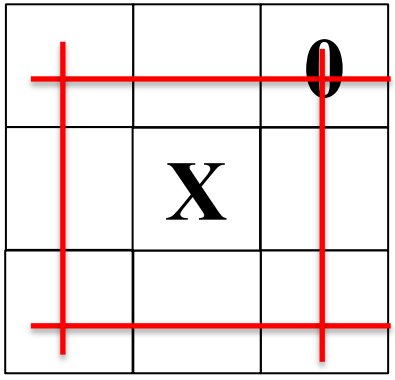
- exemplu de funcție de evaluare pentru X și 0: diferența dintre numărul de soluții disponibile pentru MAX și numărul de soluții disponibile pentru MIN
- În exemplul de mai sus: 6-4 = 2

Funcții de evaluare – X și 0

- o funcție de evaluare ideală returnează valoarea MiniMax a stării.
- diverse funcții de evaluare specifice pentru fiecare joc:





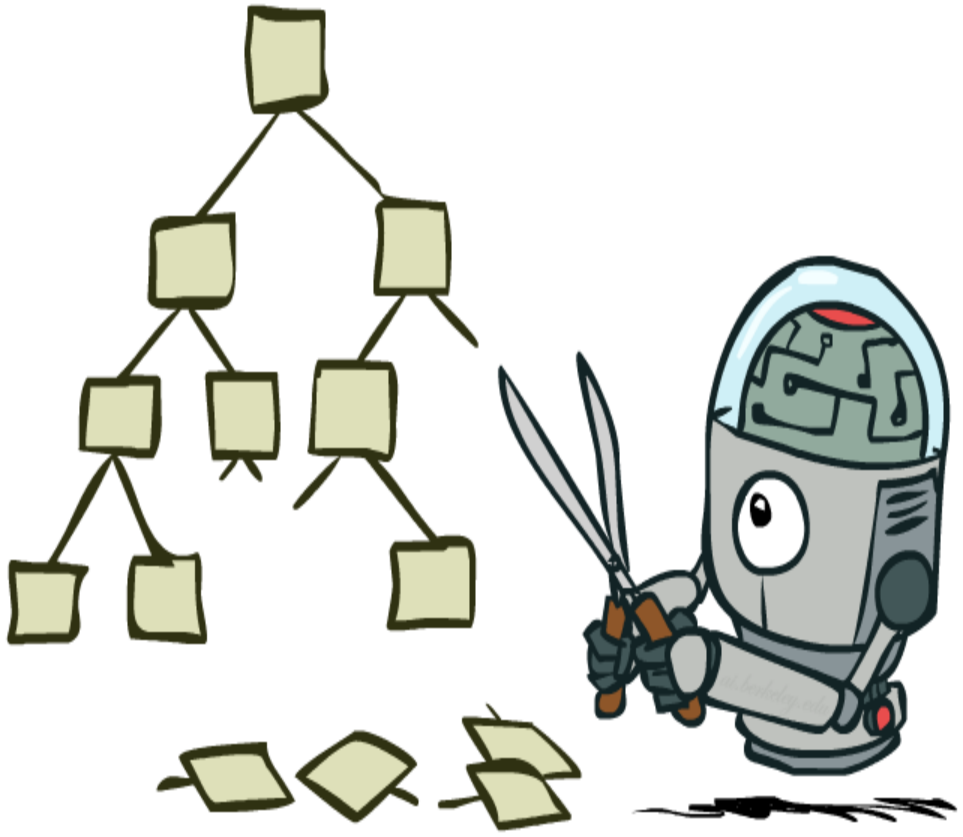


- exemplu de funcție de evaluare pentru X și 0: diferența dintre numărul de soluții disponibile pentru MAX și numărul de soluții disponibile pentru MIN
- În exemplul de mai sus: 5-4 = 1

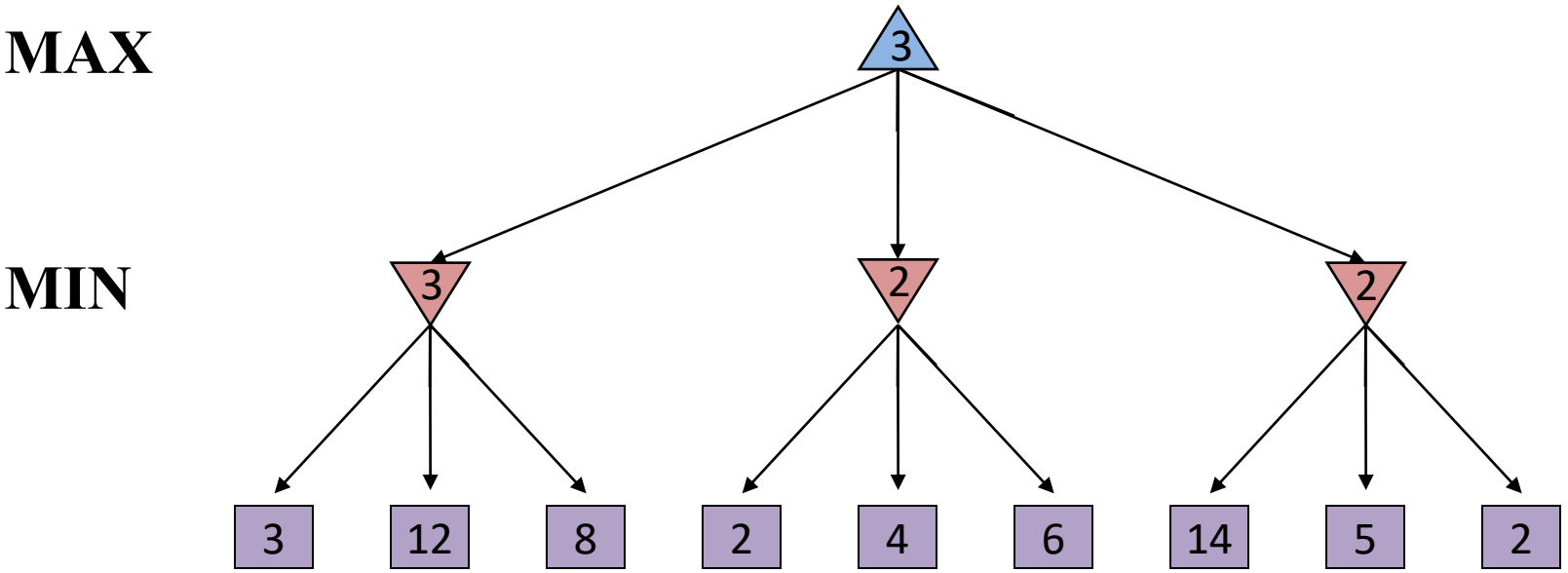
Cuprinsul cursului de azi

1. Jocuri adversariale
2. Algoritmul MINIMAX
3. **Algoritmul α-β retezare (α-β prunning)**

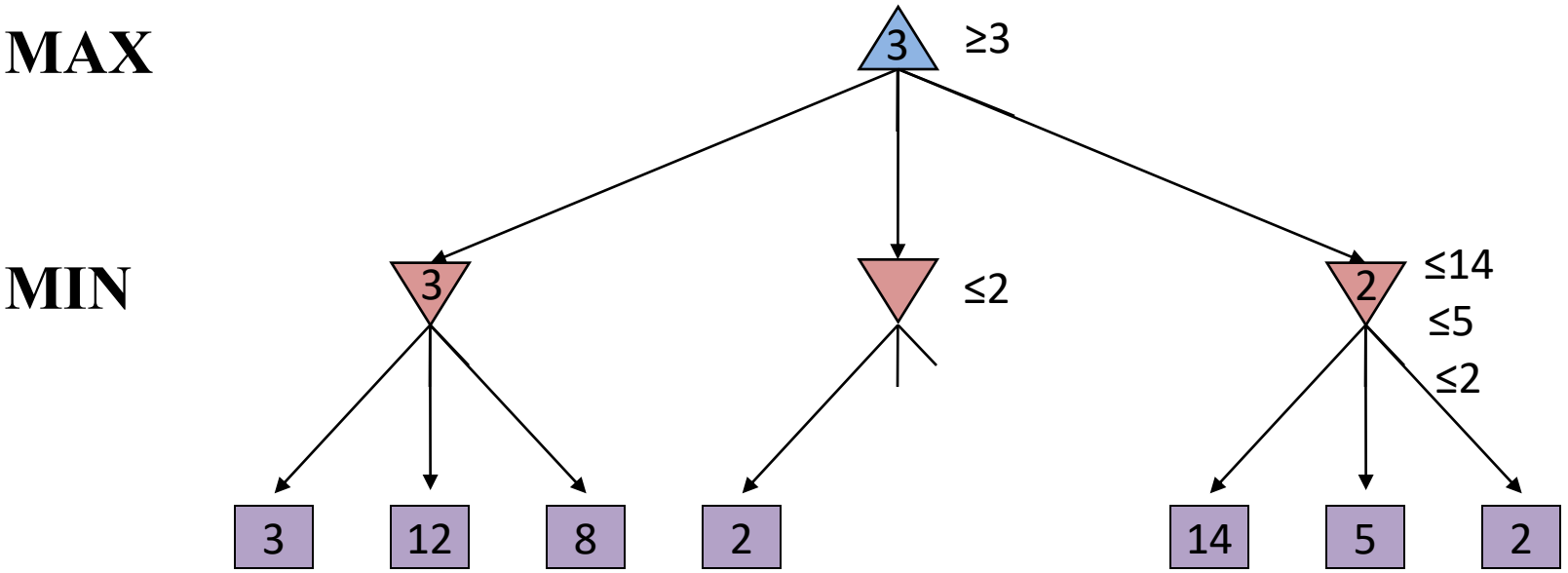
Retezări (Pruning) în arborele de joc



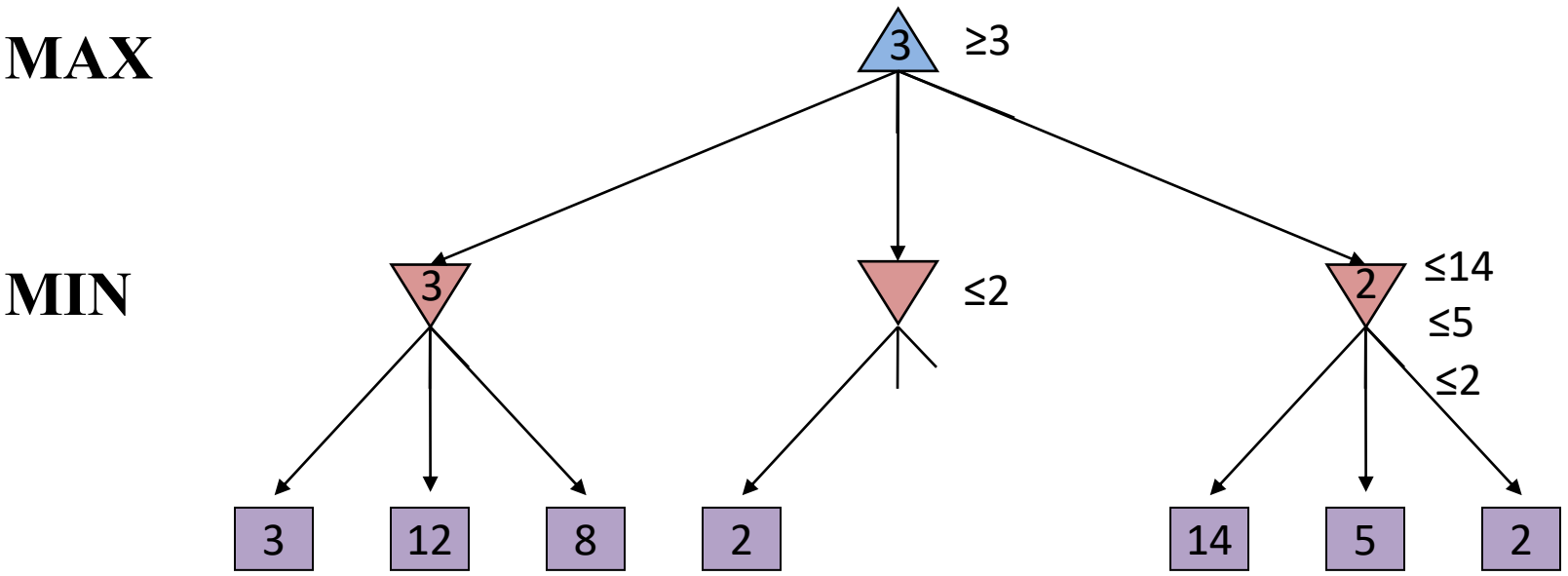
Exemplu MiniMax



Accelerarea algoritmului MiniMax



Retezare alfa-beta



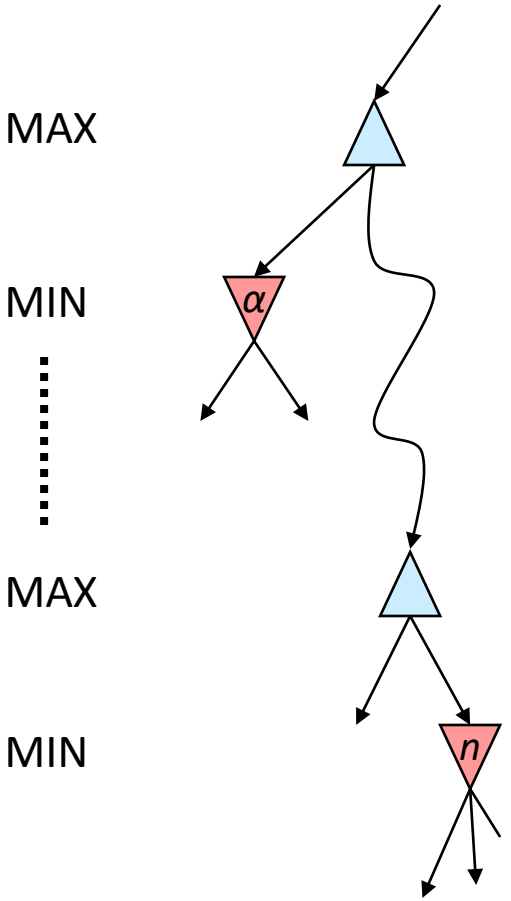
Tehnica de alfa-beta retezare, când este aplicată unui arbore de tip Minimax standard, va întoarce aceeași mutare pe care ar furniza-o și Algoritmul MiniMax, dar într-un timp mai scurt, întrucât realizează o retezare a unor ramuri (subarbori) ale arborelui care nu pot influența decizia finală și care nu mai sunt vizitate.

Principiul general al acestei tehnici constă în a considera un nod oarecare *n* al arborelui, astfel încât jucătorul poate alege să facă o mutare la acel nod. Dacă acelaşi jucător dispune de o alegere mai avantajoasă, *m*, fie la nivelul nodului părinte al lui *n*, fie în orice punct de decizie aflat mai sus în arbore, atunci *n* nu va fi niciodată atins în timpul jocului. Prin urmare, de îndată ce, în urma examinării unora dintre descendenţii nodului *n*, ajungem să deţinem suficientă informaţie relativ la acesta, îl putem înlătura.

Retezare alfa-beta (alfa-beta pruning)

- Cazurile când se aplică (pentru noduri de tip MIN)
  - calculăm pentru jucătorul MIN valoarea nodului *n*
  - iterăm după toți succesorii nodului *n* de la stânga la dreapta
  - valoarea estimată a nodului *n* va scădea pe măsură ce îi vizităm toți succesorii
  - cine este interesat de valoarea nodului *n*? MAX
  - fie α cea mai bună valoare curentă (garantată) pe care MAX o poate obține după ce a vizitat subarborii din stânga subarborelui actual vizitat pornind de la rădăcină
  - dacă valoarea curentă estimată a lui *n* devine mai mică decât α, MAX nu va alege acest drum, deci putem să renunțăm (să retezăm) să evaluăm valorile pentru celelalte noduri succesor ale lui *n*.

- Cazurile când se aplică (pentru noduri de tip MAX)
  - simetric, se definește în mod similar valoarea β ca fiind cea mai bună valoare (garantată) pe care MIN o poate obține.
  - MIN nu mai trebuie să mai ia în considerare nicio valoare internă mai mare sau egală cu β care este asociată oricărui nod intern *n* de tip MAX. Putem tăia (reteza) acea ramură a arborelui de joc.



- **Ideea tehnicii de alpha-beta retezare: a găsi o mutare "suficient de bună", nu neapărat cea mai bună, dar suficient de bună pentru a se lua decizia corectă. Această idee poate fi formalizată prin introducerea a *două limite*, *alpha* şi *beta*, reprezentând limitări ale valorii de tip minimax corespunzătoare unui nod intern.**

- **Semnificaţia acestor limite este următoarea:**

  - ***alpha* este *valoarea minimă* pe care este deja garantat că o va obţine MAX;**
  - ***beta* este *valoarea maximă* pe care MAX poate spera să o atingă.**

- **Din punctul de vedere al jucătorului MIN, beta este valoarea cea mai nefavorabilă pentru MIN pe care acesta o va atinge.**

- ***Valoarea efectivă* care va fi găsită se află *între alpha şi beta*.**

Valoarea alpha, asociată nodurilor de tip MAX, nu poate niciodată să descrească, iar valoarea beta, asociată nodurilor de tip MIN, nu poate niciodată să crească.

- **Alpha este scorul cel mai prost pe care îl poate obţine MAX, presupunând că MIN joacă perfect.**

Dacă, spre exemplu, valoarea alpha a unui nod intern de tip MAX este 6, atunci MAX nu mai trebuie să ia în consideraţie nici o valoare internă mai mică sau egală cu 6 care este asociată oricărui nod de tip MIN situat sub el. În mod similar, dacă MIN are valoarea beta 6, el nu mai trebuie să ia în consideraţie nici un nod de tip MAX situat sub el care are valoarea 6 sau o valoare mai mare decât acest număr.

- **Cele două reguli pentru încheierea căutării, bazată pe valori alpha şi beta, sunt:**

1. **Căutarea poate fi oprită dedesubtul oricărui nod de tip MIN care are o valoare beta mai mică sau egală cu valoarea alpha a oricăruia dintre strămoşii săi de tip MAX.**

2. **Căutarea poate fi oprită dedesubtul oricărui nod de tip MAX care are o valoare alpha mai mare sau egală cu valoarea beta a oricăruia dintre strămoşii săi de tip MIN.**

Implementare Alfa-Beta retezare

α: opțiunea curentă cea mai bună a lui MAX (MAX vrea să maximizeze α)
β: opțiunea curentă cea mai bună a lui MIN (MIN vrea să minimizeze β)
Întotdeauna vom avea $\alpha \leq \beta$

```
def max-value(stare, α, β):   //nodul n e de tip MAX, MAX are α, MIN are β
    initializeaza v = -∞
    pentru fiecare succesor al stării:          //pentru fiecare succesor al lui n
        v = max(v, value(successor, α, β))   //calculăm valoarea v și actualizăm max
        if v ≥ β return v    //dacă v depășește limita β a lui MIN nu mai continuăm
                             // întoarcem în acest caz v  = o estimare a lui n
        α = max(α, v)        // actualizăm valoarea α
    return v
```

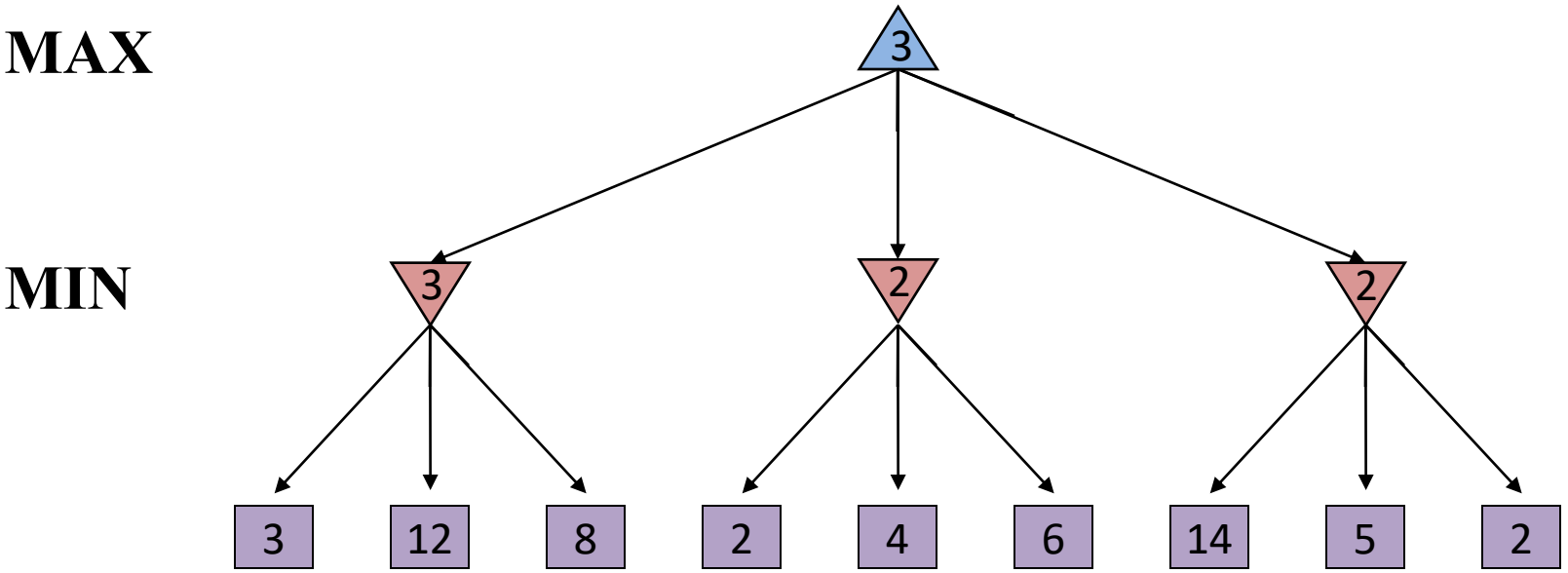
Implementare Alfa-Beta retezare

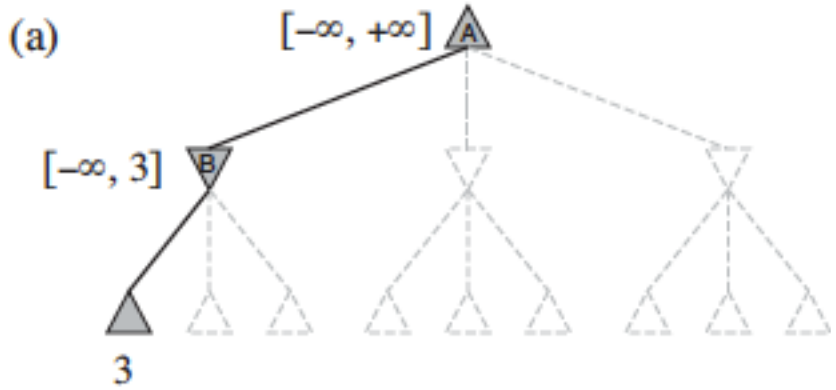
α: opțiunea curentă cea mai bună a lui MAX (MAX vrea să maximizeze α)
β: opțiunea curentă cea mai bună a lui MIN (MIN vrea să minimizeze β)
Întotdeauna vom avea $\alpha \leq \beta$

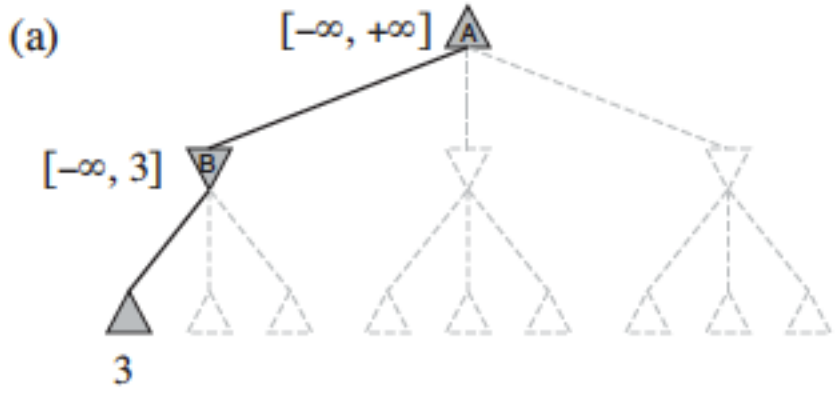
```
def max-value(stare, α, β):
    initializeaza v = -∞
    pentru fiecare succesor al stării:
        v = max(v, value(successor, α, β))
        if v ≥ β return v
        α = max(α, v)
    return v
```

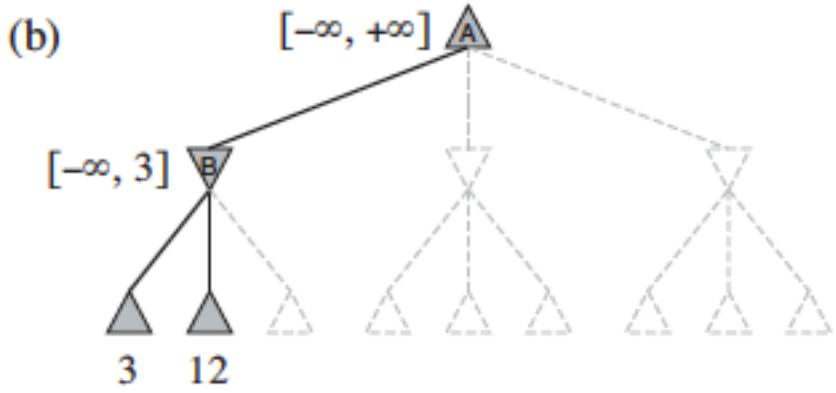
```
def min-value(stare , α, β):
    initializeaza v = +∞
    pentru fiecare succesor al stării:
        v = min(v, value(successor, α, β))
        if v ≤ α return v
        β = min(β, v)
    return v
```

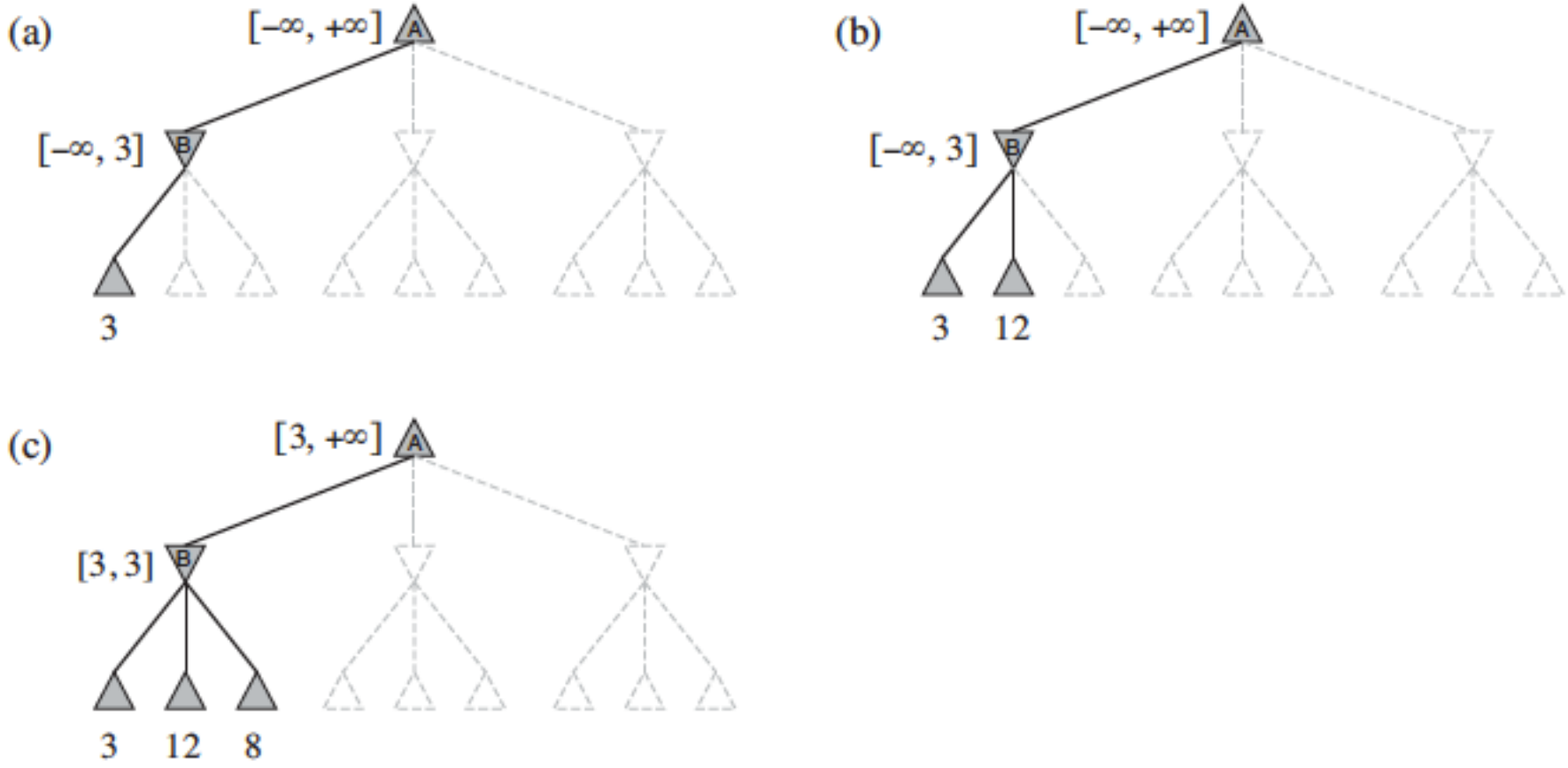
Exemplu MiniMax

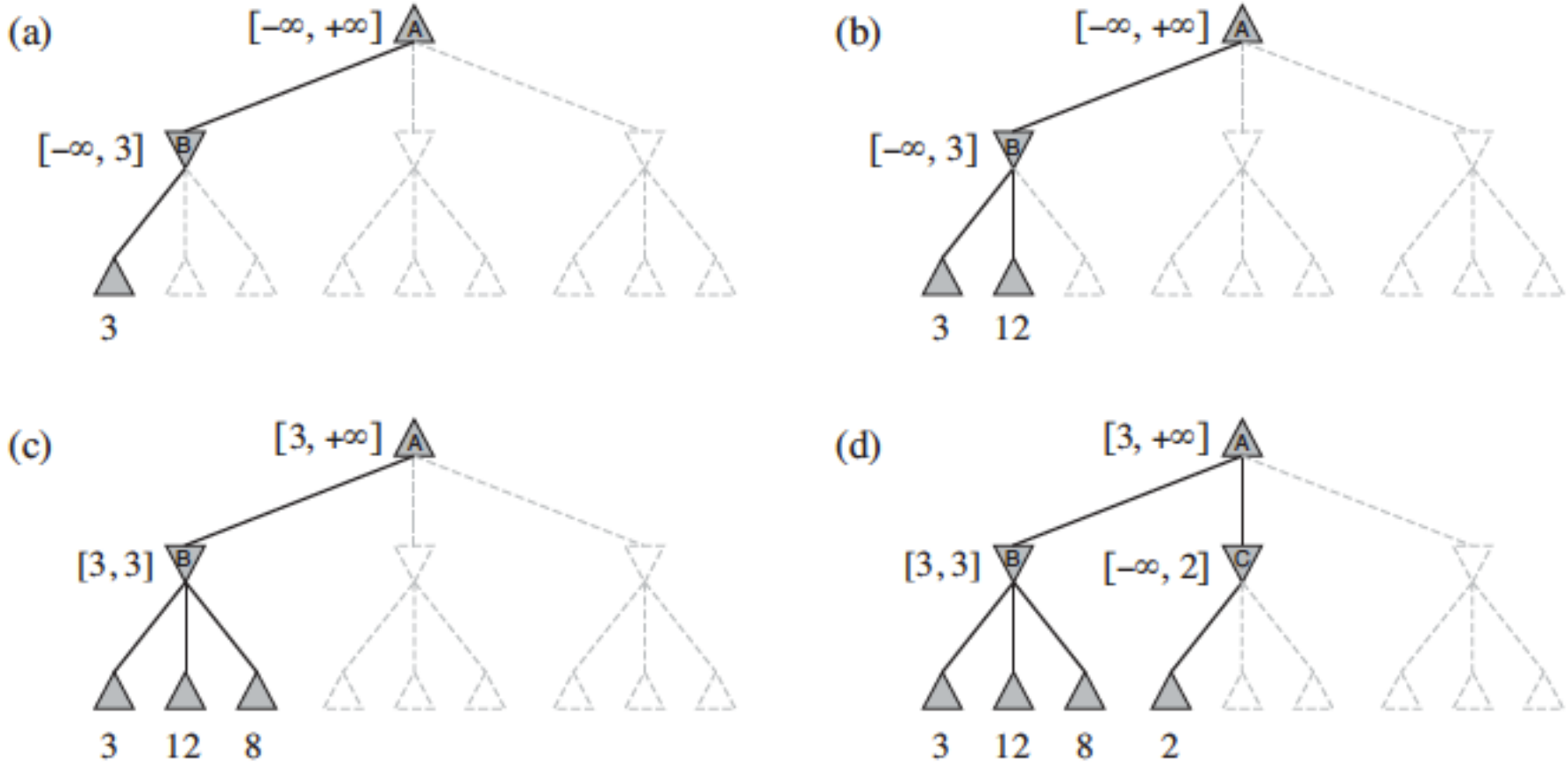


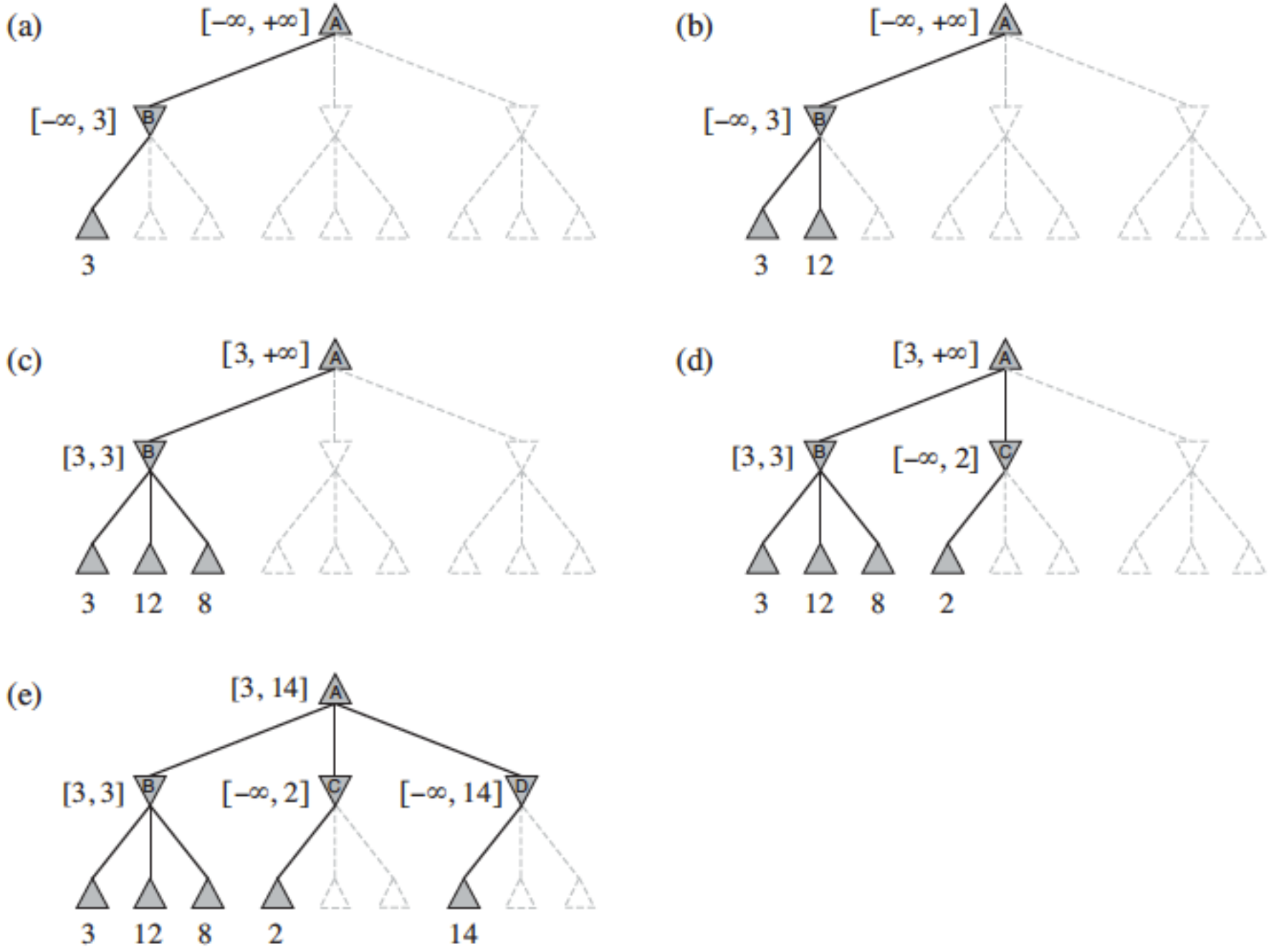

(a)
[−∞, +∞]
A
[−∞, 3]
B
3

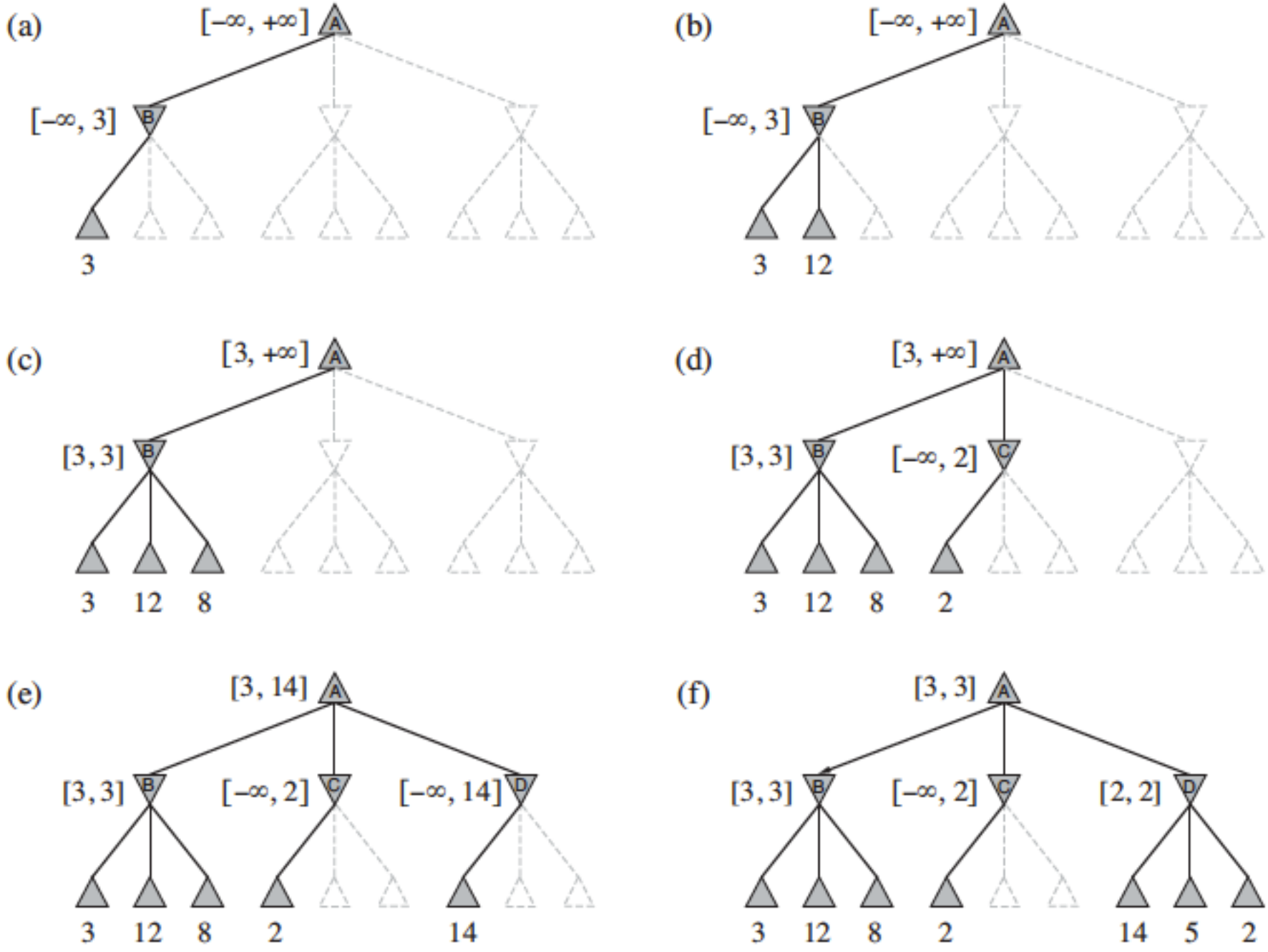

(a)
[−∞, +∞]
A
[−∞, 3]
B
3


(b)
[−∞, +∞]
A
[−∞, 3]
B
3
12

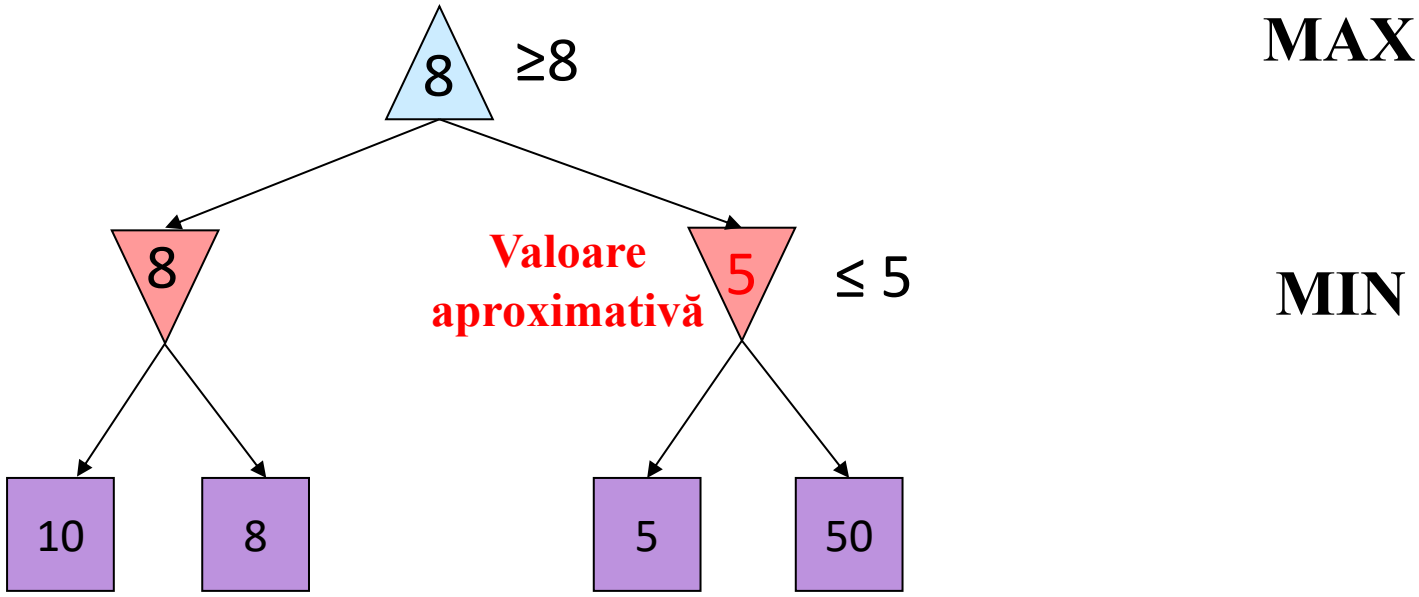

(a)
[−∞, +∞]
A
[−∞, 3]
B
3
(b)
[−∞, +∞]
A
[−∞, 3]
B
3
12
(c)
[3, +∞]
A
[3, 3]
B
3
12
8


(a)
[−∞, +∞]
A
[−∞, 3]
B
3
(b)
[−∞, +∞]
A
[−∞, 3]
B
3
12
(c)
[3, +∞]
A
[3, 3]
B
3
12
8
(d)
[3, +∞]
A
[3, 3]
B
[−∞, 2]
C
3
12
8
2

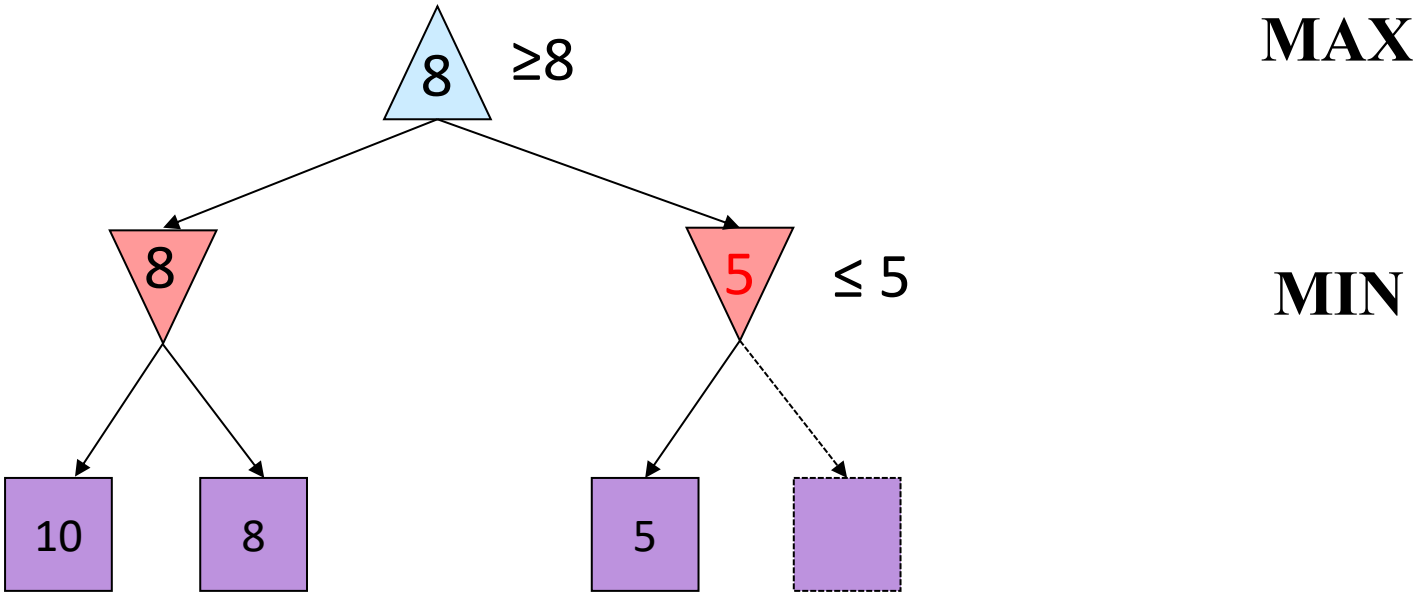

(a)
[−∞, +∞]
A
[−∞, 3]
B
3
(b)
[−∞, +∞]
A
[−∞, 3]
B
3
12
(c)
[3, +∞]
A
[3, 3]
B
3
12
8
(d)
[3, +∞]
A
[3, 3]
B
[−∞, 2]
C
3
12
8
2
(e)
[3, 14]
A
[3, 3]
B
[−∞, 2]
C
[−∞, 14]
D
3
12
8
2
14


(a)
[−∞, +∞]
A
[−∞, 3]
B
3
(b)
[−∞, +∞]
A
[−∞, 3]
B
3
12
(c)
[3, +∞]
A
[3, 3]
B
3
12
8
(d)
[3, +∞]
A
[3, 3]
B
[−∞, 2]
C
3
12
8
2
(e)
[3, 14]
A
[3, 3]
B
[−∞, 2]
C
[−∞, 14]
D
3
12
8
2
14
(f)
[3, 3]
A
[3, 3]
B
[−∞, 2]
C
[2, 2]
D
3
12
8
2
14
5
2

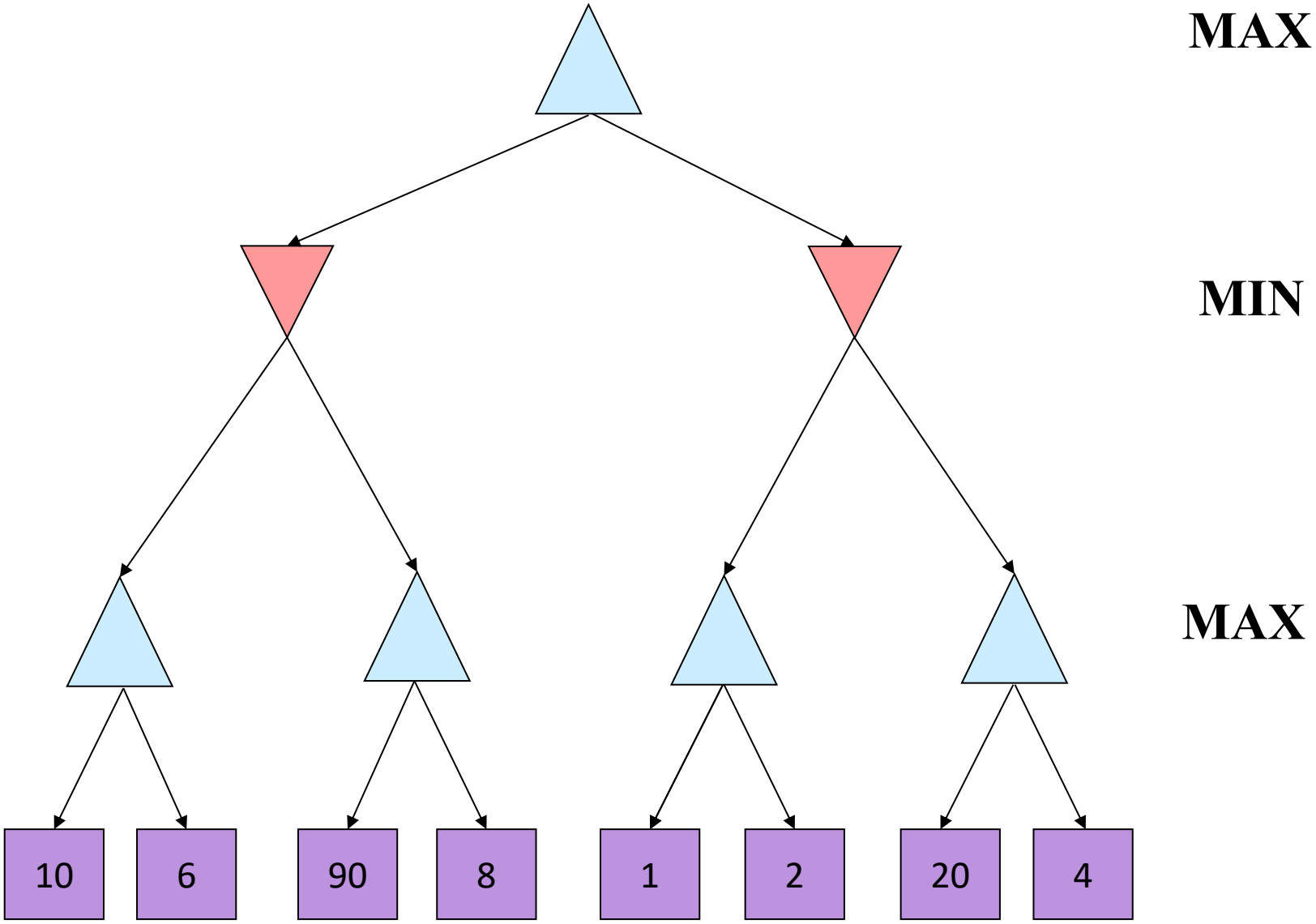
Exemplu



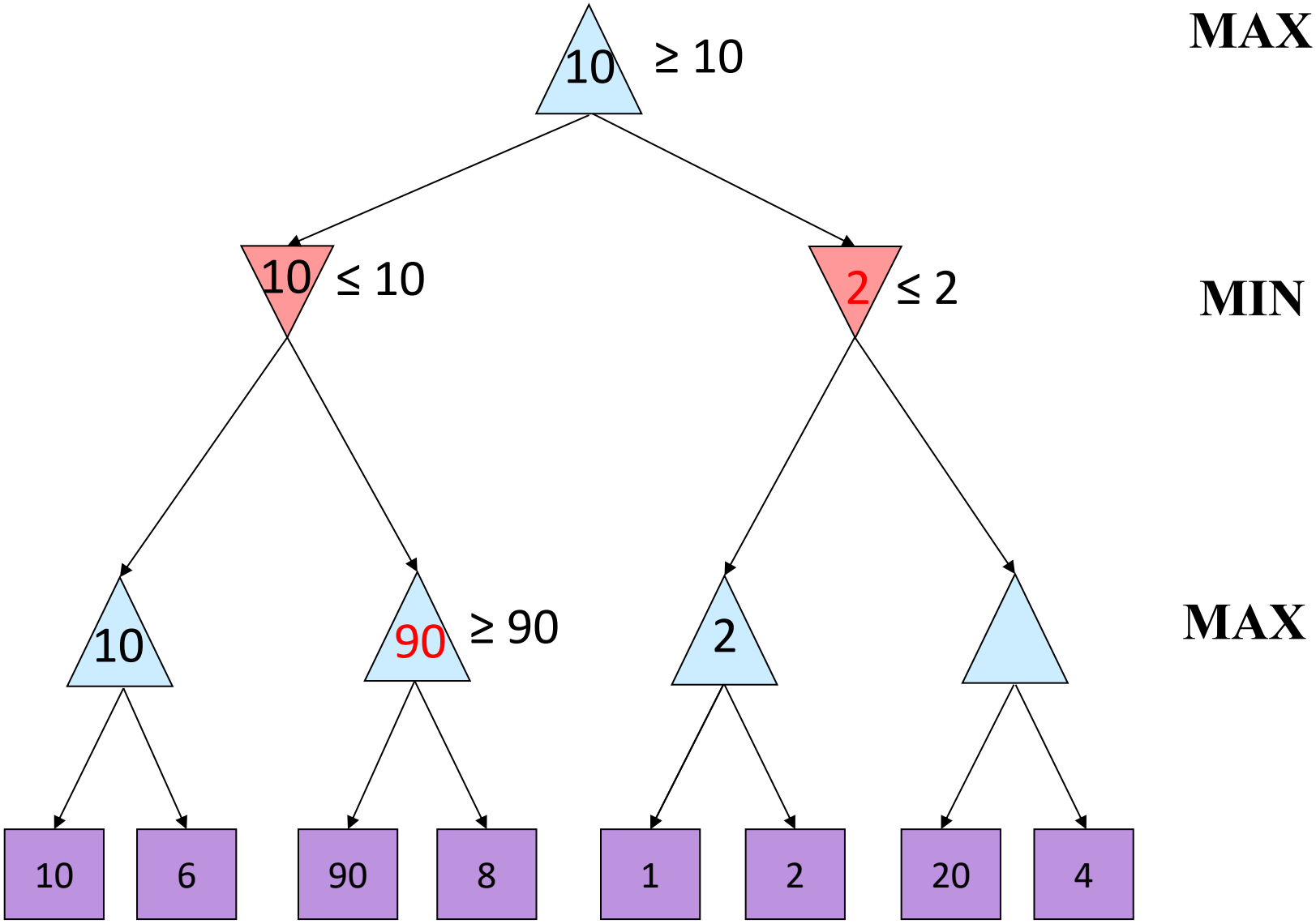
Exemplu



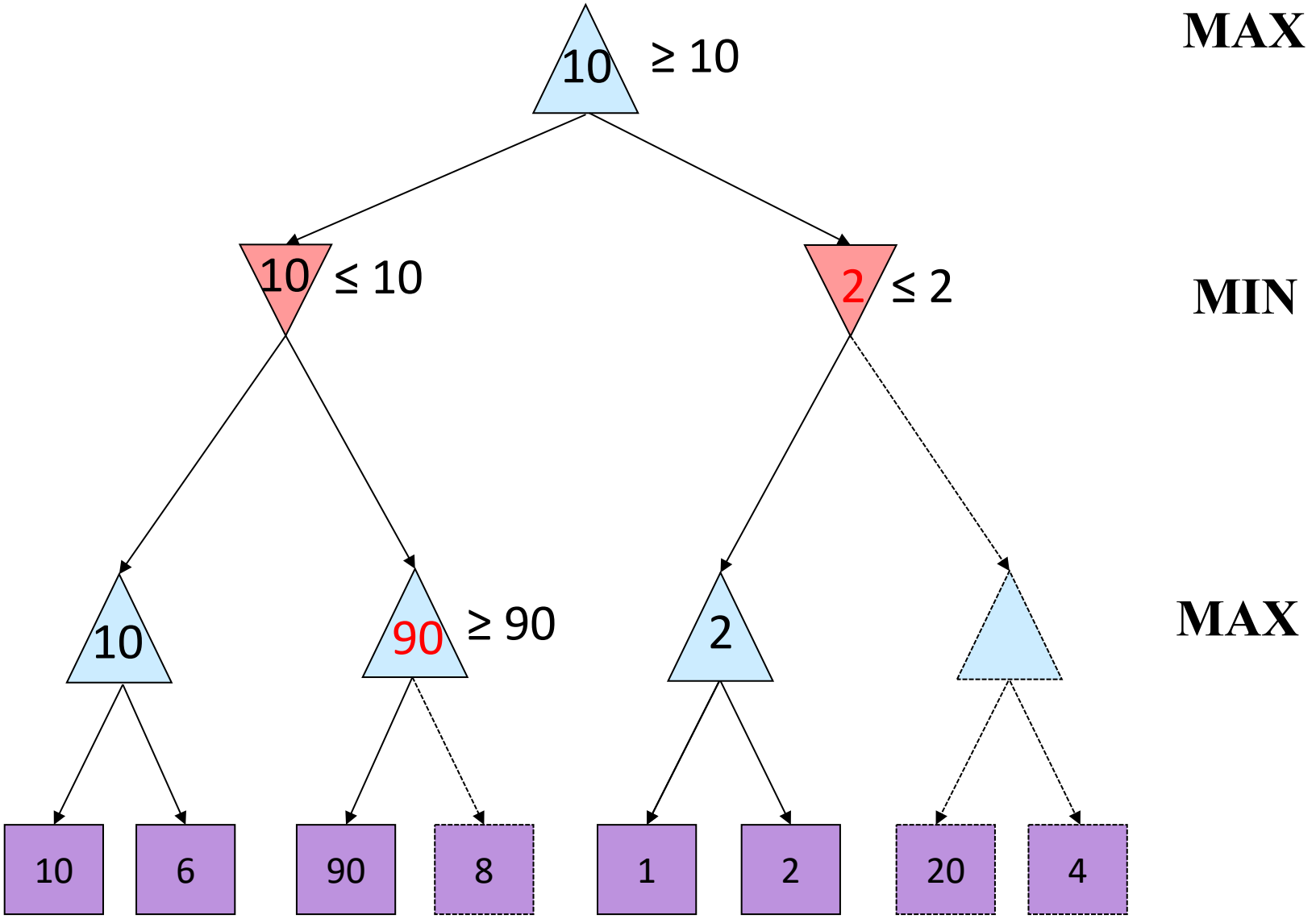
Exemplu



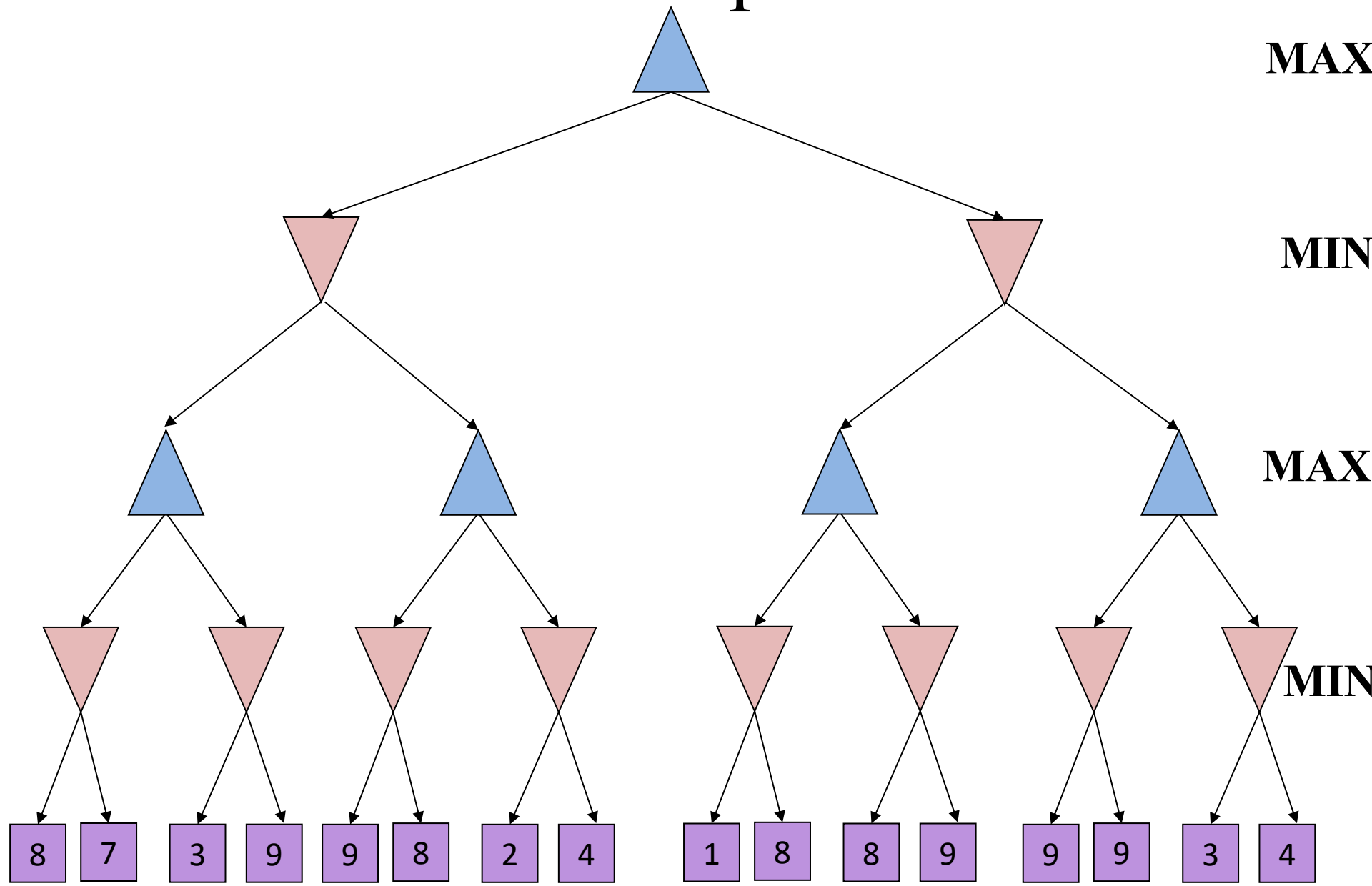
Exemplu



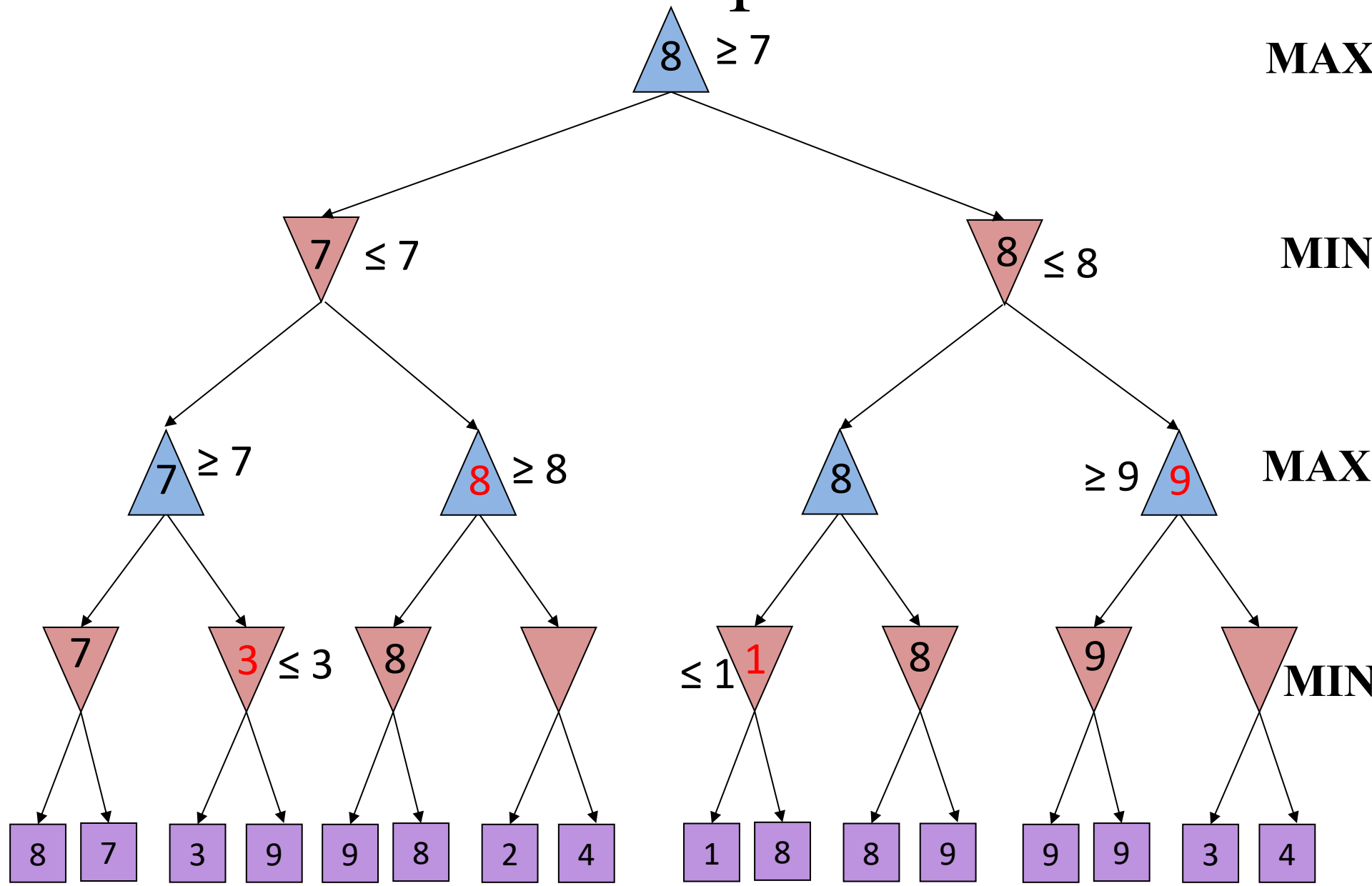
Exemplu



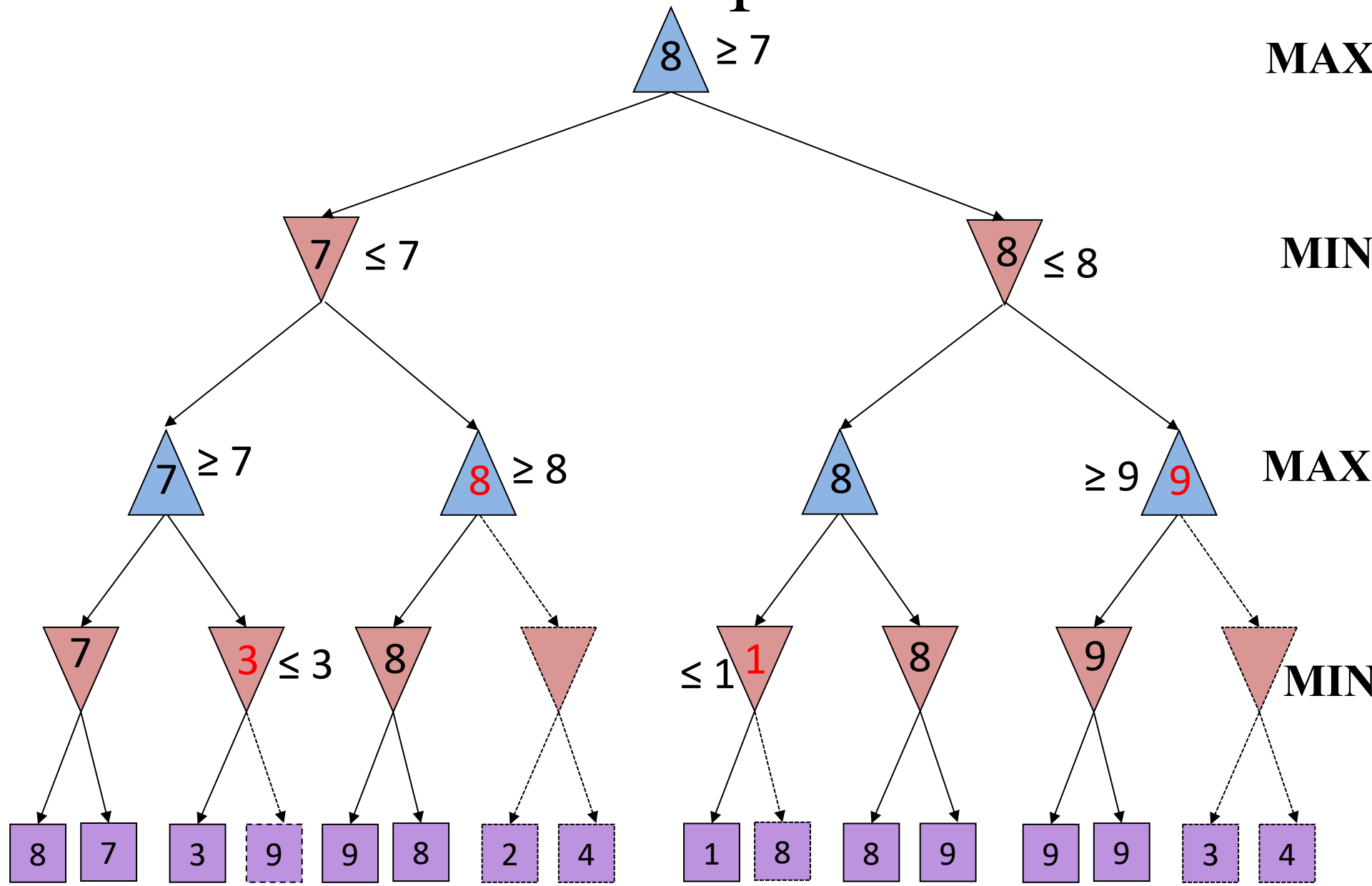
Exemplu



Exemplu



Exemplu



Proprietățile algoritmului Alpha-Beta Pruning

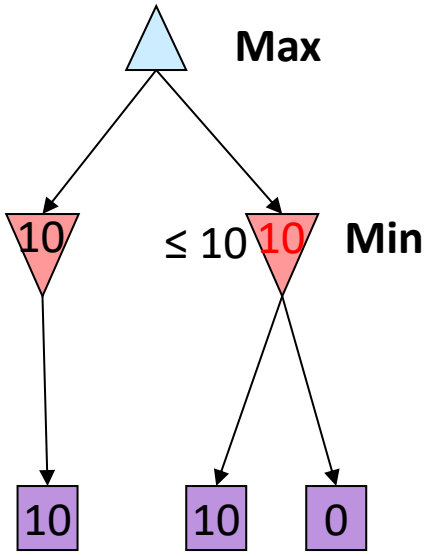
- procesul de retezare nu are niciun efect asupra valorii minimax calculate pentru nodul rădăcină (aceasta nu se schimbă)

- valorile nodurilor intermediare (aproximate) pot fi greșite:
  - important: nodurile fii ale rădăcinii pot avea valori greșite

  Implementarea naivă poate conduce la erori

- vizitarea în ordinea bună a nodurilor fii conduce la retezări mai rapide. Este preferabil să fie examinaţi mai întâi succesorii despre care se crede că ar putea fi cei mai buni

- cu "ordonare perfectă":
  - complexitatea timp poate scădea la $O(b^{m/2})$
  - algoritmul poate rula pe o adâncime dublă
  - căutarea exhuastivă nu este totuși posibilă…



Considerații privitoare la eficiență

Eficienţa Algoritmului Alpha-Beta depinde de *ordinea în care sunt examinaţi succesorii.* Este preferabil să fie examinaţi mai întâi succesorii despre care se crede că ar putea fi cei mai buni. În mod evident, acest lucru nu poate fi realizat în întregime. Dacă el ar fi posibil, funcţia care ordonează succesorii ar putea fi utilizată pentru a se juca un joc perfect. În ipoteza în care această ordonare ar putea fi realizată, s-a arătat că Algoritmul Alpha-Beta nu trebuie să examineze, pentru a alege cea mai bună mutare, decât $O(b^{d/2})$ noduri, în loc de $O(b^d)$, ca în cazul Algoritmului Minimax. Aceasta arată că factorul de ramificare efectiv este $\sqrt{b}$ în loc de b – în cazul jocului de şah 6, în loc de 35. Cu alte cuvinte, Algoritmul Alpha-Beta poate "privi înainte" la o adâncime dublă faţă de Algoritmul Minimax, pentru a găsi acelaşi cost.

În cazul unei ordonări neprevăzute a stărilor în spaţiul de căutare, Algoritmul Alpha-Beta poate dubla adâncimea spaţiului de căutare (Nilsson 1980). Dacă există o anumită ordonare nefavorabilă a nodurilor, acest algoritm nu va căuta mai mult decât Algoritmul Minimax. Prin urmare, în cazul cel mai nefavorabil, Algoritmul Alpha-Beta nu va oferi nici un avantaj în comparaţie cu căutarea exhaustivă de tip minimax. În cazul unei ordonări favorabile însă, dacă notăm prin N numărul poziţiilor de căutare terminale evaluate în mod static de către Algoritmul Minimax, s-a arătat că, în cazul cel mai bun, adică atunci când mutarea cea mai puternică este prima luată în consideraţie, Algoritmul Alpha-Beta nu va evalua în mod static decât $\sqrt{N}$ poziţii. În practică, o funcţie de ordonare relativ simplă (cum ar fi încercarea mai întâi a capturilor, apoi a ameninţărilor, apoi a mutărilor înainte, apoi a celor înapoi) ne poate apropia suficient de mult de rezultatul obţinut în cazul cel mai favorabil.

Subiect dat în trecut la examenul la CTI

Subiectul 4. (2 puncte)

În jocul Conectează-3, jucătorii *X* şi 0 mută alternativ plasând simbolurile lor într-una din coloanele 1, 2, 3 sau 4 ale unui tablou cu 3 linii şi 4 coloane. Simbolurile se acumulează unul peste altul (Figura 3a). O coloană în care au fost plasate 3 simboluri devine plină şi prin urmare jucătorii nu mai pot plasa simboluri în ea. Câştigă jucătorul care realizează primul 3 simboluri dispuse pe orizontală, verticală sau diagonală. Dacă niciun jucător nu realizează acest lucru jocul se termină la egalitate. Jucătorul *X* mută primul.

(a) Care este numărul de mutări maxim (= factorul de ramificare) pe care îl are la dispoziţie fiecare jucător în orice moment al jocului? Justificaţi răspunsul. **(0,25 puncte)**

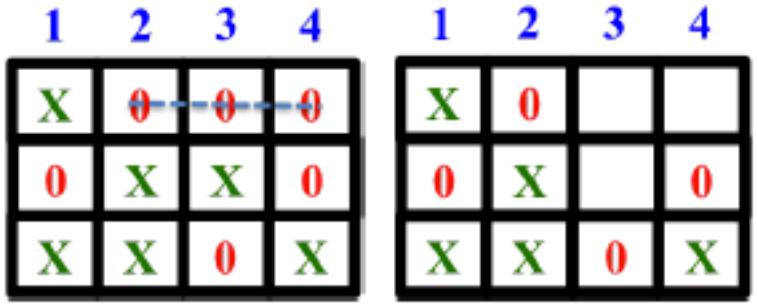


Figura 3: a. Exemplu de stare finală (nod de tip frunză în arborele de joc) cu utilitatea −1 (câştigă jucătorul 0 ce realizează o linie pe orizontală); b. Stare curentă în care se ajunge după 9 mutări, acum jucătorul 0 este la mutare.

Subiectul 4a – rezolvare

Subiectul 4

a) Numărul maxim de mutări (= factorul de ramificare) pe care îl are la dispoziție fiecare jucător = numărul de coloane libere. Inițial, toate coloanele sunt libere, deci $b = 4$, spre sfârșitul jocului se poate întâmpla ca numai o singură coloană să fie liberă, deci $b = 1$.

Subiectul 4b – enunț

Subiectul 4. (2 puncte)

În jocul Conectează-3, jucătorii *X* şi 0 mută alternativ plasând simbolurile lor într-una din coloanele 1, 2, 3 sau 4 ale unui tablou cu 3 linii şi 4 coloane. Simbolurile se acumulează unul peste altul (Figura 3a). O coloană în care au fost plasate 3 simboluri devine plină şi prin urmare jucătorii nu mai pot plasa simboluri în ea. Câştigă jucătorul care realizează primul 3 simboluri dispuse pe orizontală, verticală sau diagonală. Dacă niciun jucător nu realizează acest lucru jocul se termină la egalitate. Jucătorul *X* mută primul.

(b) Considerăm că după 9 mutări ajungem cu jocul în starea dată de Figura 3b. Jucătorul 0 este acum la mutare. Desenaţi arborele de joc asociat, considerând drept rădăcină starea actuală a jocului. **(0,75 puncte)**

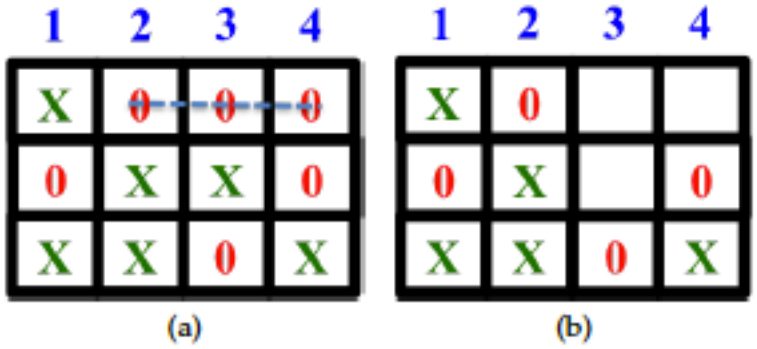
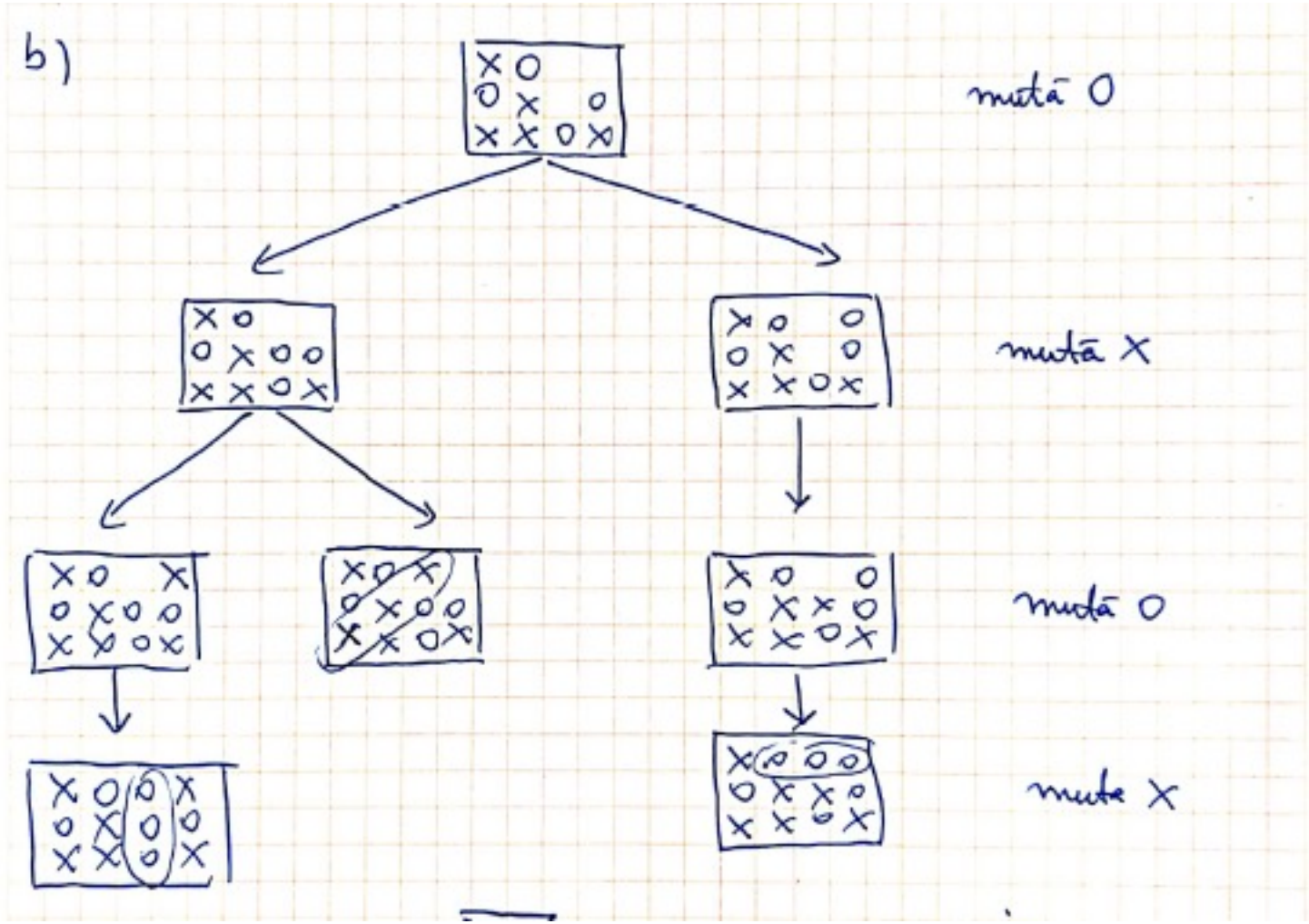


Figura 3: a. Exemplu de stare finală (nod de tip frunză în arborele de joc) cu utilitatea −1 (câştigă jucătorul 0 ce realizează o linie pe orizontală); b. Stare curentă în care se ajunge după 9 mutări, acum jucătorul 0 este la mutare.

Subiectul 4b – rezolvare



Subiectul 4c – enunț

Subiectul 4. (2 puncte)

În jocul Conectează-3, jucătorii X şi 0 mută alternativ plasând simbolurile lor într-una din coloanele 1, 2, 3 sau 4 ale unui tablou cu 3 linii şi 4 coloane. Simbolurile se acumulează unul peste altul (Figura 3a). O coloană în care au fost plasate 3 simboluri devine plină şi prin urmare jucătorii nu mai pot plasa simboluri în ea. Câştigă jucătorul care realizează primul 3 simboluri dispuse pe orizontală, verticală sau diagonală. Dacă niciun jucător nu realizează acest lucru jocul se termină la egalitate. Jucătorul X mută primul.

(c) considerăm jucătorul X ca fiind MAX şi jucătorul 0 ca fiind MIN. Funcţia de utilitate asociată nodurilor terminale este următoarea: dacă jucătorul X câştigă atunci el primeşte k puncte iar jucătorul 0 pierde k puncte, dacă jocul se termină la egalitate atunci fiecare are 0 puncte. Valoarea k atunci când jucătorul X câştigă este stabilită astfel: $k = 3$ puncte pentru o linie realizată din simboluri dispuse pe diagonală, $k = 2$ pentru o linie realizată din simboluri dispuse pe verticală şi $k = 1$ pentru o linie realizată din simboluri dispuse pe orizontală. Valoarea k atunci când jucătorul 0 câştigă pe cazurile enumerate anterior este $-3, -2, -1$. Folosind algoritmul Minimax etichetaţi arborele de joc de la punctul anterior cu valorile minimax asociate fiecărui nod. **(0,5 puncte)**

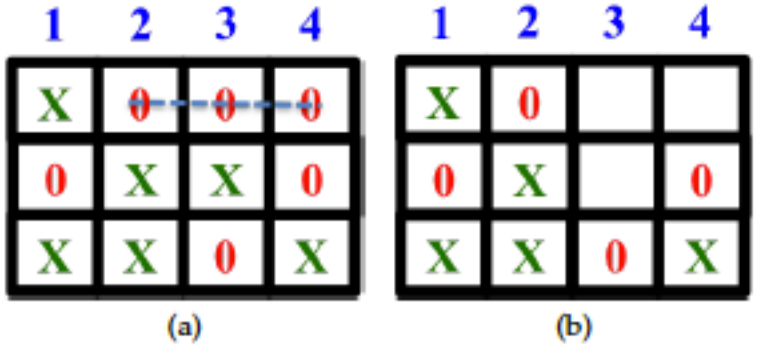
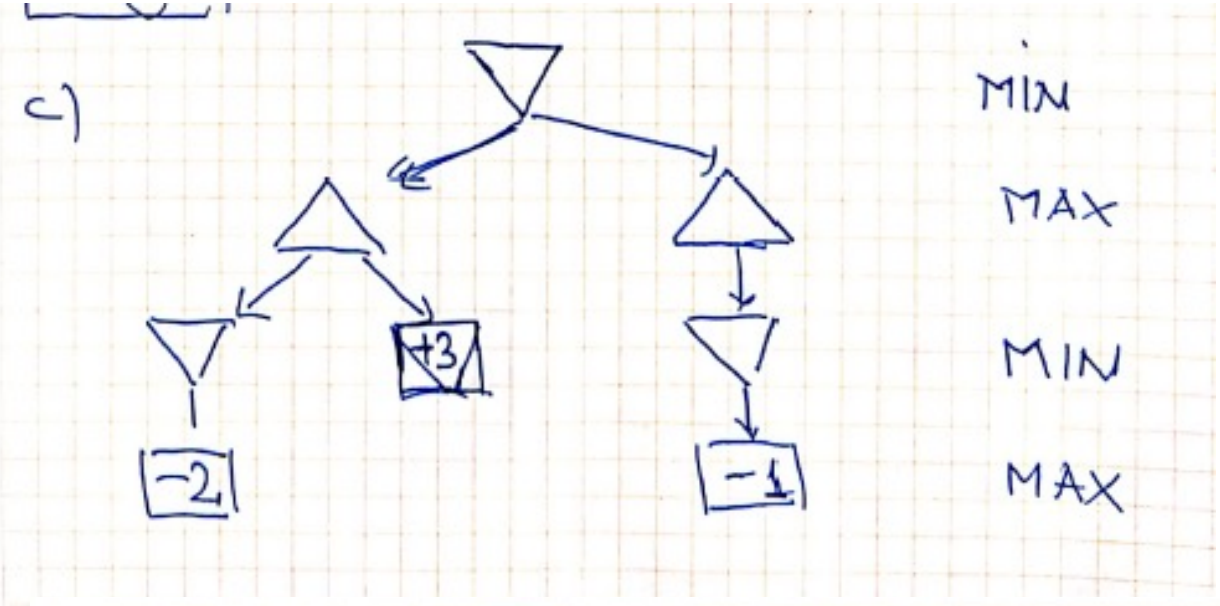
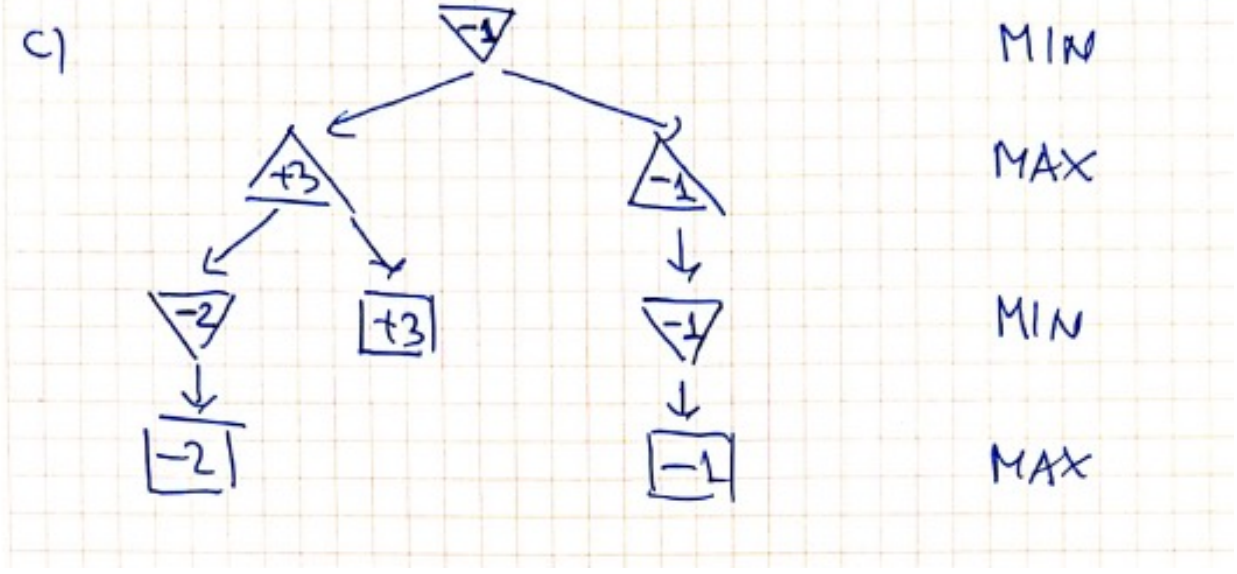


Figura 3: a. Exemplu de stare finală (nod de tip frunză în arborele de joc) cu utilitatea -1 (câştigă jucătorul 0 ce realizează o linie pe orizontală); b. Stare curentă în care se ajunge după 9 mutări, acum jucătorul 0 este la mutare.

Subiectul 4c – rezolvare





Subiectul 4d – enunț

Subiectul 4. (2 puncte)

În jocul Conectează-3, jucătorii *X* şi 0 mută alternativ plasând simbolurile lor într-una din coloanele 1, 2, 3 sau 4 ale unui tablou cu 3 linii şi 4 coloane. Simbolurile se acumulează unul peste altul (Figura 3a). O coloană în care au fost plasate 3 simboluri devine plină şi prin urmare jucătorii nu mai pot plasa simboluri în ea. Câştigă jucătorul care realizează primul 3 simboluri dispuse pe orizontală, verticală sau diagonală. Dacă niciun jucător nu realizează acest lucru jocul se termină la egalitate. Jucătorul *X* mută primul.

(d) Folosiţi algoritmul de retezare Alfa-Beta pentru accelerarea calculului valorilor Minimax. Ce noduri din arbore de joc nu vor fi explorate? Puteţi reoordona fiii fiecărui subarbore pentru a maximiza numărul de noduri care nu vor fi explorate de algoritmul Alfa-Beta? Justificaţi răspunsul. **(0,5 puncte)**

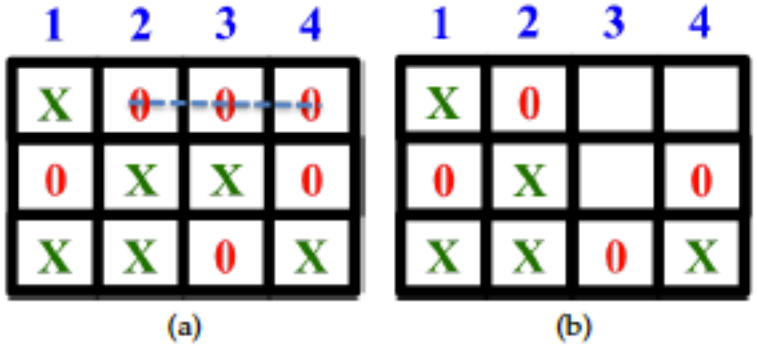
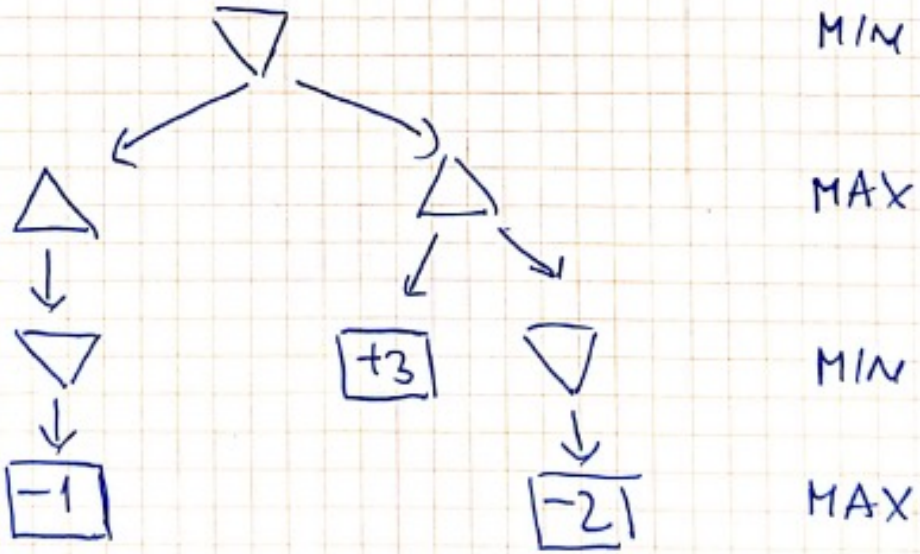


Figura 3: a. Exemplu de stare finală (nod de tip frunză în arborele de joc) cu utilitatea −1 (câştigă jucătorul 0 ce realizează o linie pe orizontală); b. Stare curentă în care se ajunge după 9 mutări, acum jucătorul 0 este la mutare.

d] Cu Alfa-Beta, în configurația actuală nu se poate face nicio retezare.

Rearanjăm arborele astfel:



Aplicăm Alfa Beta