



Mesterséges intelligencia előadássorozat

Az előadás diái az AIMA könyvre épülve (<http://aima.cs.berkeley.edu>) készültek a University of California, Berkeley mesterséges intelligencia kurzusának anyagainak felhasználásával (<http://ai.berkeley.edu>).

These slides are based on the AIMA book (<http://aima.cs.berkeley.edu>) and were adapted from the AI course material of University of California, Berkeley (<http://ai.berkeley.edu>).



Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Mesterséges Intelligencia és Rendszertervezés Tanszék



Mesterséges intelligencia

Keresés ellenséges környezetben és a játékok

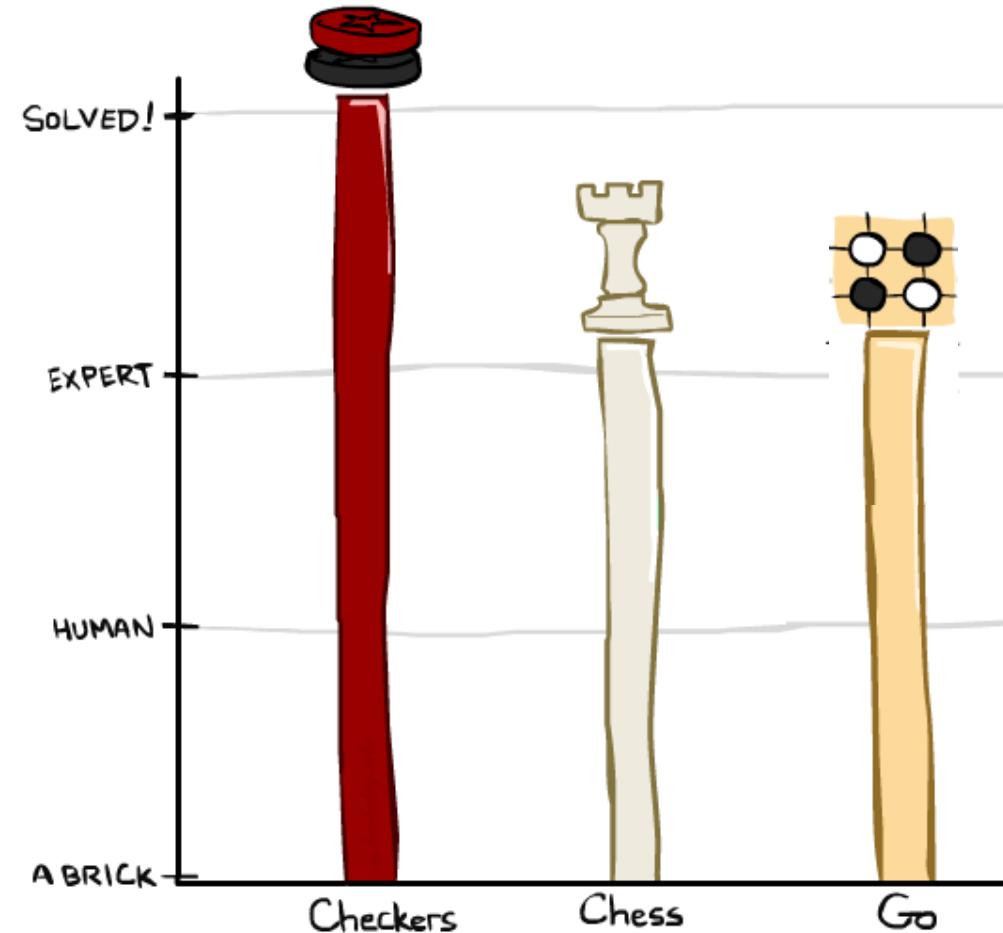
Előadó: Dr. Hullám Gábor

Előadás anyaga: Dr. Gézsi András
Dr. Hullám Gábor



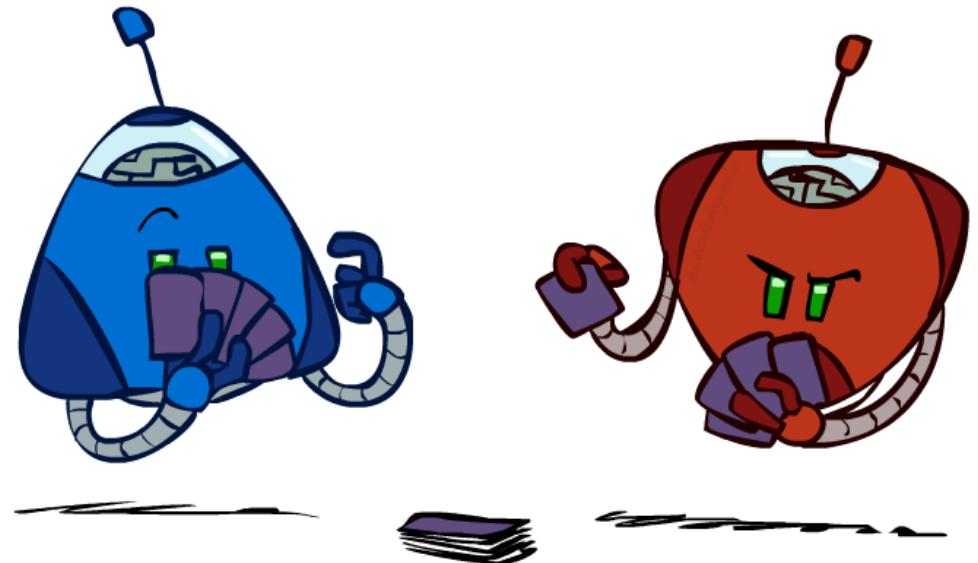
Játékok – a létező legjobb MI játékosok

- **Dámajáték:** 1950: Első számítógépes játékos. 1994: Első számítógépes bajnok: Chinook törte meg az emberi bajnok, Marion Tinsley 40 évig tartó bajnoki címét, majd Tinsley visszavonulása után átvette a bajnoki címet. 2007: Dámajáték megoldása!
- **Sakk:** 1997: Deep Blue legyőzte az emberi bajnokot, Gary Kasparovot egy hat játékból álló meccsben. Deep Blue 200 millió pozíciót vizsgált meg másodpercenként, rendkívül szofisztikált kiértékelést és egyéb nem nyilvános módszereket használt, amelyekkel a keresést akár 40 lépés mélysegig is ki tudta terjeszteni. A jelenlegi programok még jobbak (pl. Stockfish, AlphaZero).
- **Go:** 2015-ig amatőr szint, 2016: AlphaGO legyőzte az emberi bajnokot. 2018: AlphaZero legyőzte AlphaGO-t (Monte Carlo fa keresés (MCTS), megerősítéses tanulás, neurális hálók, önmagával való játék)



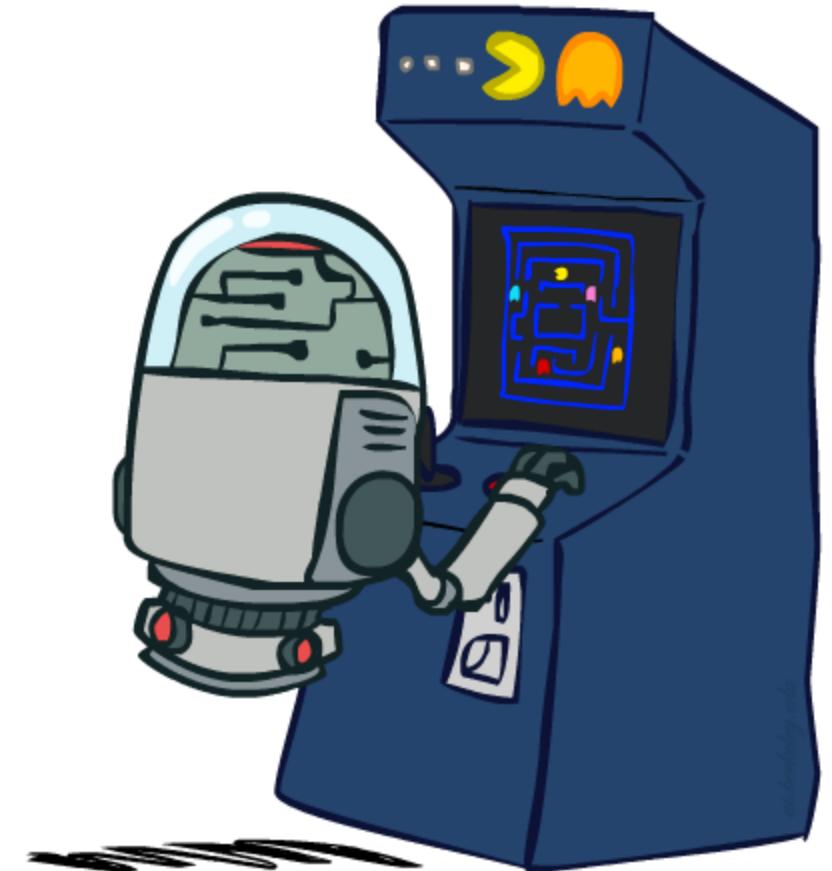
Játékok típusai

- Rendkívül sokféle játék létezik
- Tulajdonságok:
 - Determinisztikus vagy sztochasziktus?
 - Egy, kettő vagy több játékos?
 - Zéró összegű játék?
 - Teljes információ (láthatjuk az állapotot)?
- Algoritmus = **stratégia**t ad optimális/jó választ javasol minden egyes állapotban

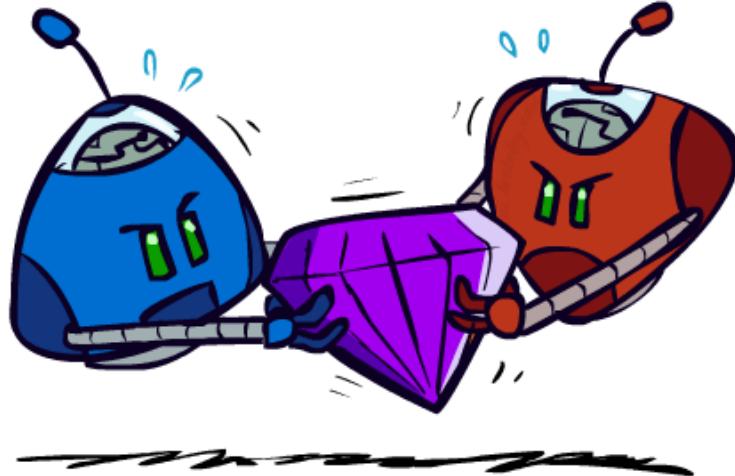


Determinisztikus játékok

- Egy lehetséges formalizmus:
 - Állapotok: S (s_0 kezdőállapot)
 - Játékosok: $P=\{1\dots N\}$ (rendszerint felváltva lépnek)
 - Cselekvések: A (a soron következő játékos és az aktuális állapot szabja meg)
 - Állapotátmenet függvény: $S \times A \rightarrow S$
 - Célállapot teszt: $S \rightarrow \{\text{igaz}, \text{hamis}\}$
 - Célállapot hasznosság: $S \times P \rightarrow R$
- A megoldás egy játékos számára a **stratégia**: $S \rightarrow A$



Zéró összegű játékok



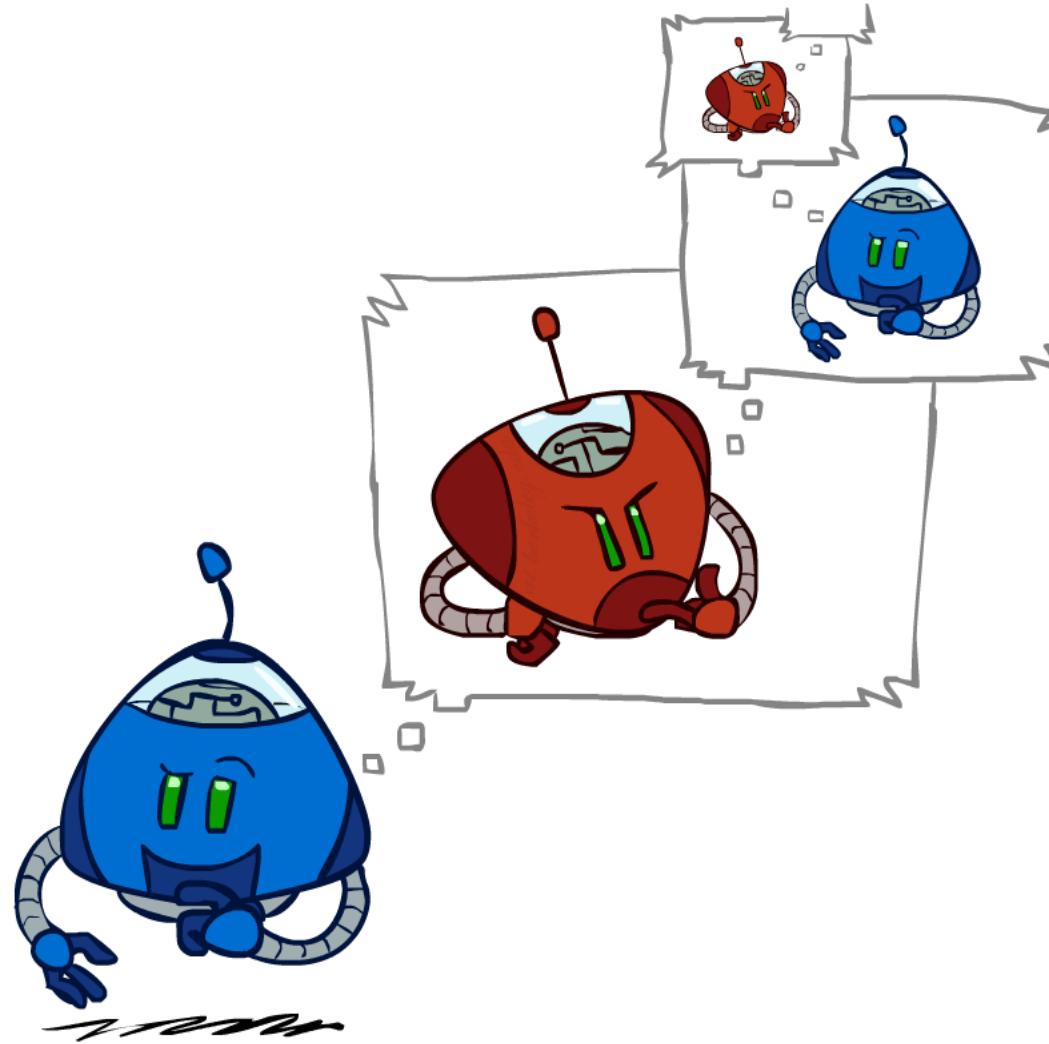
■ Zéró összegű játékok

- Az ágensek hasznossága (a kimenetelek értéke) ellenkező
- Egyedüli értékként is tekinthető, amelyet az egyik játékos maximalizálni akar, a másik pedig minimalizálni
- Ellenséges játékos, tiszta versengés

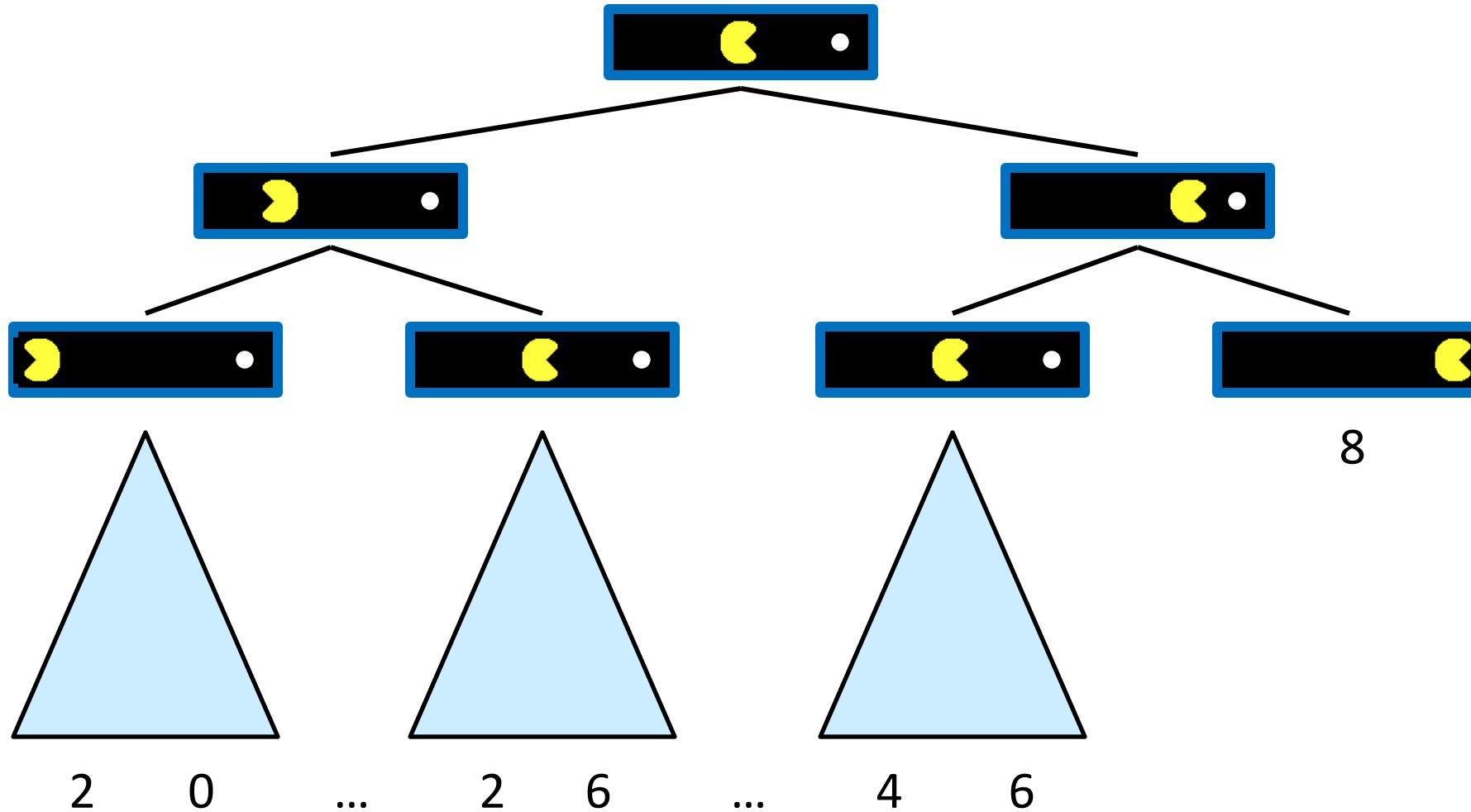
■ Általános (nem zéró összegű) játékok

- Az ágensek hasznossága (a kimenetelek értéke) független
- Kooperáció, közömbösségek, versengés stb. mind elképzelhető

Keresés ellenséges környezetben

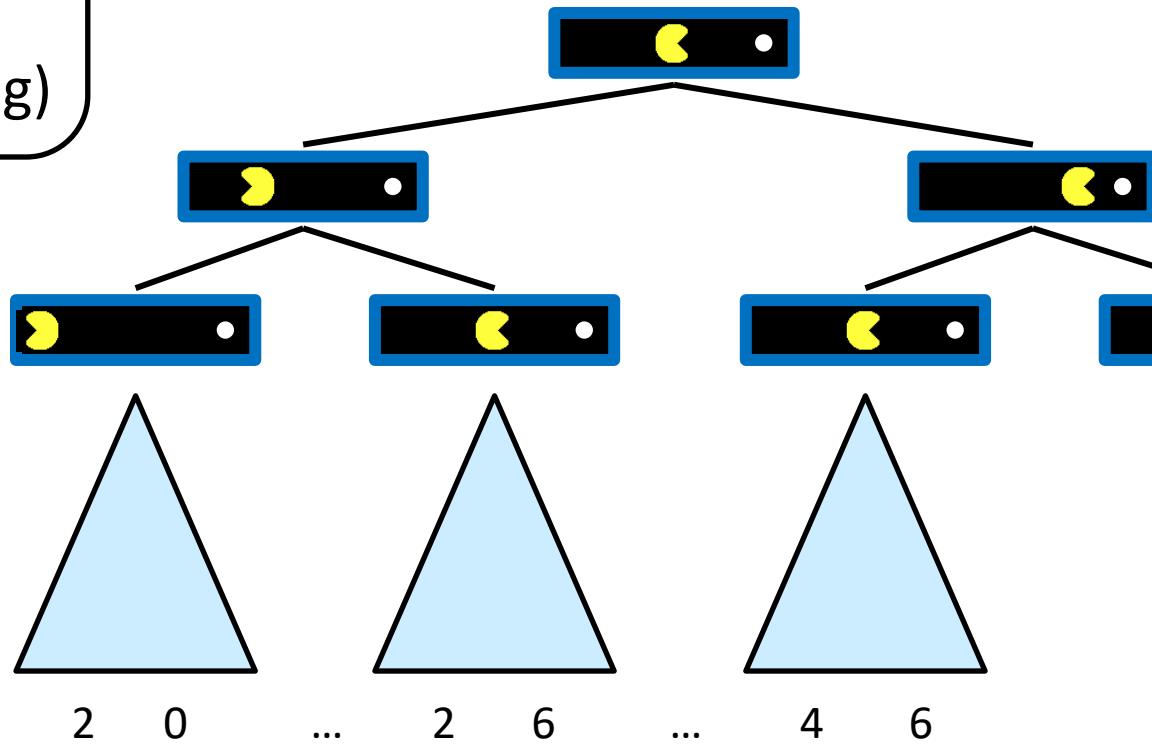


Egy-ágenses játékfa



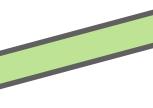
Egy állapot értéke

Egy állapot értéke:
Az adott állapotból
elérhető legjobb
kimenetel (hasznosság)

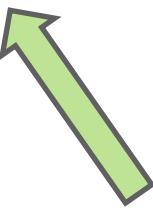


Nem végállapotok:

$$V(s) = \max_{s' \in \text{követő-állapot}(s)} V(s')$$



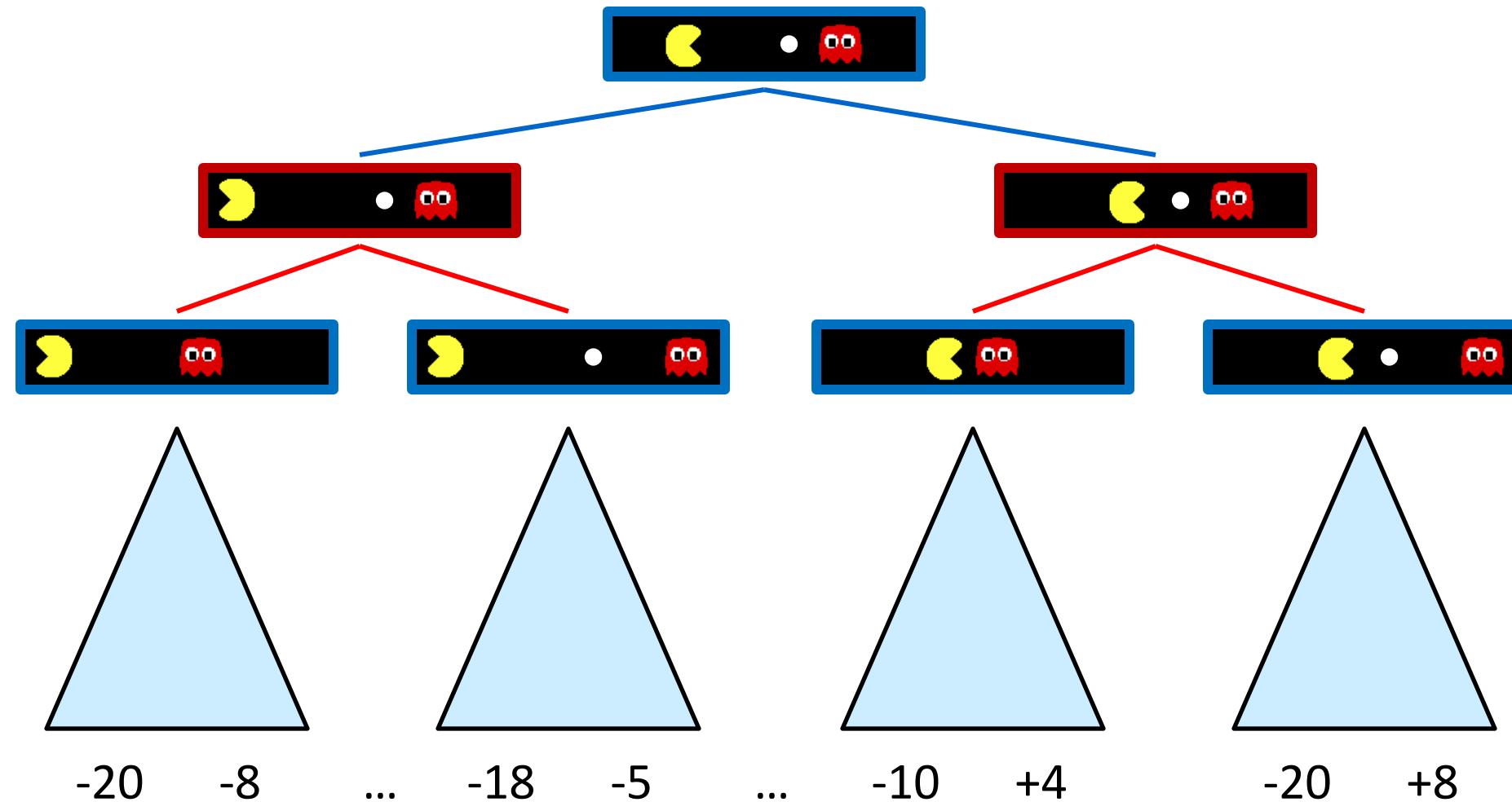
8



Végállapotok:

$$V(s) = \text{ismert}$$

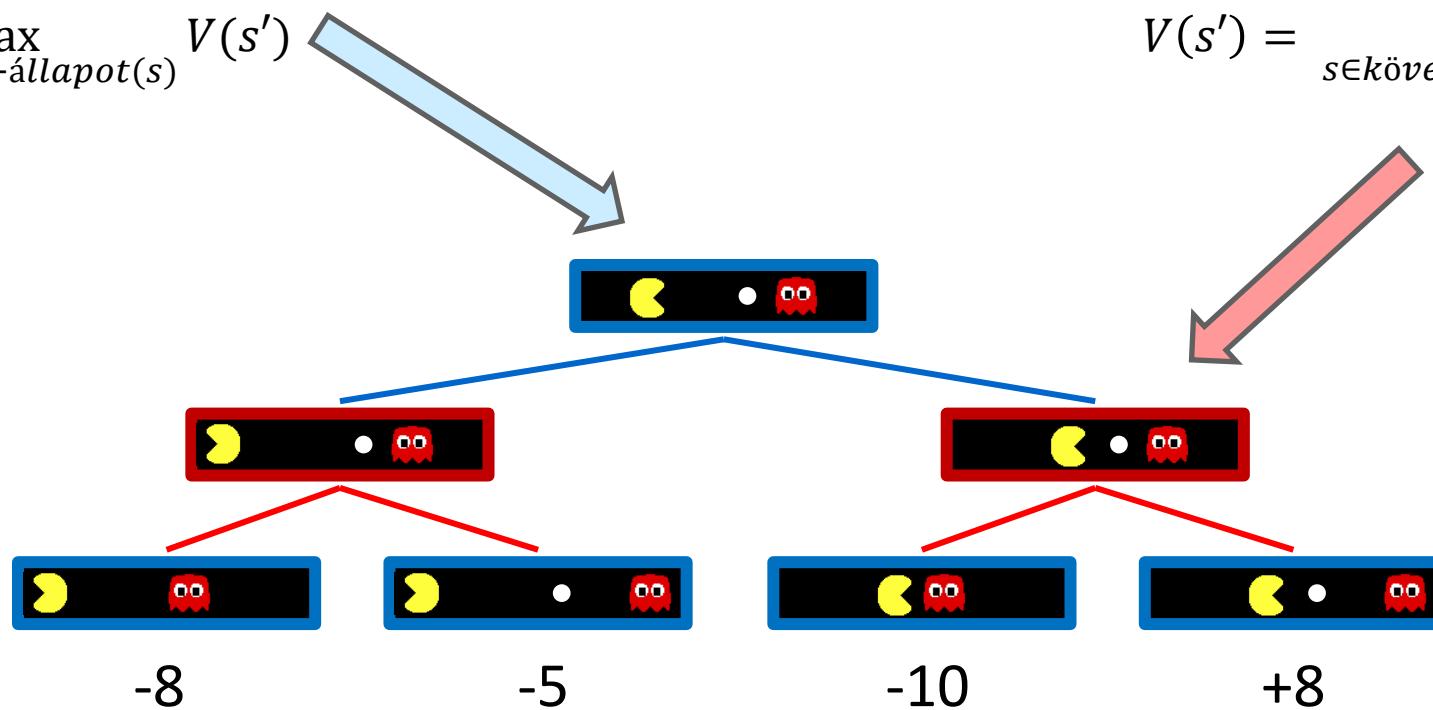
Két-ágenses játékfa (Ellenség melletti keresés)



Minimax értékek

Az ágens által kontrollált állapotok:

$$V(s) = \max_{s' \in \text{követő-állapot}(s)} V(s')$$

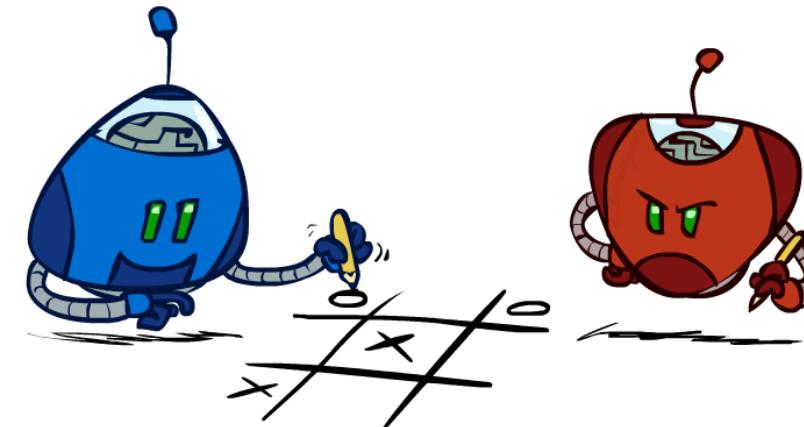
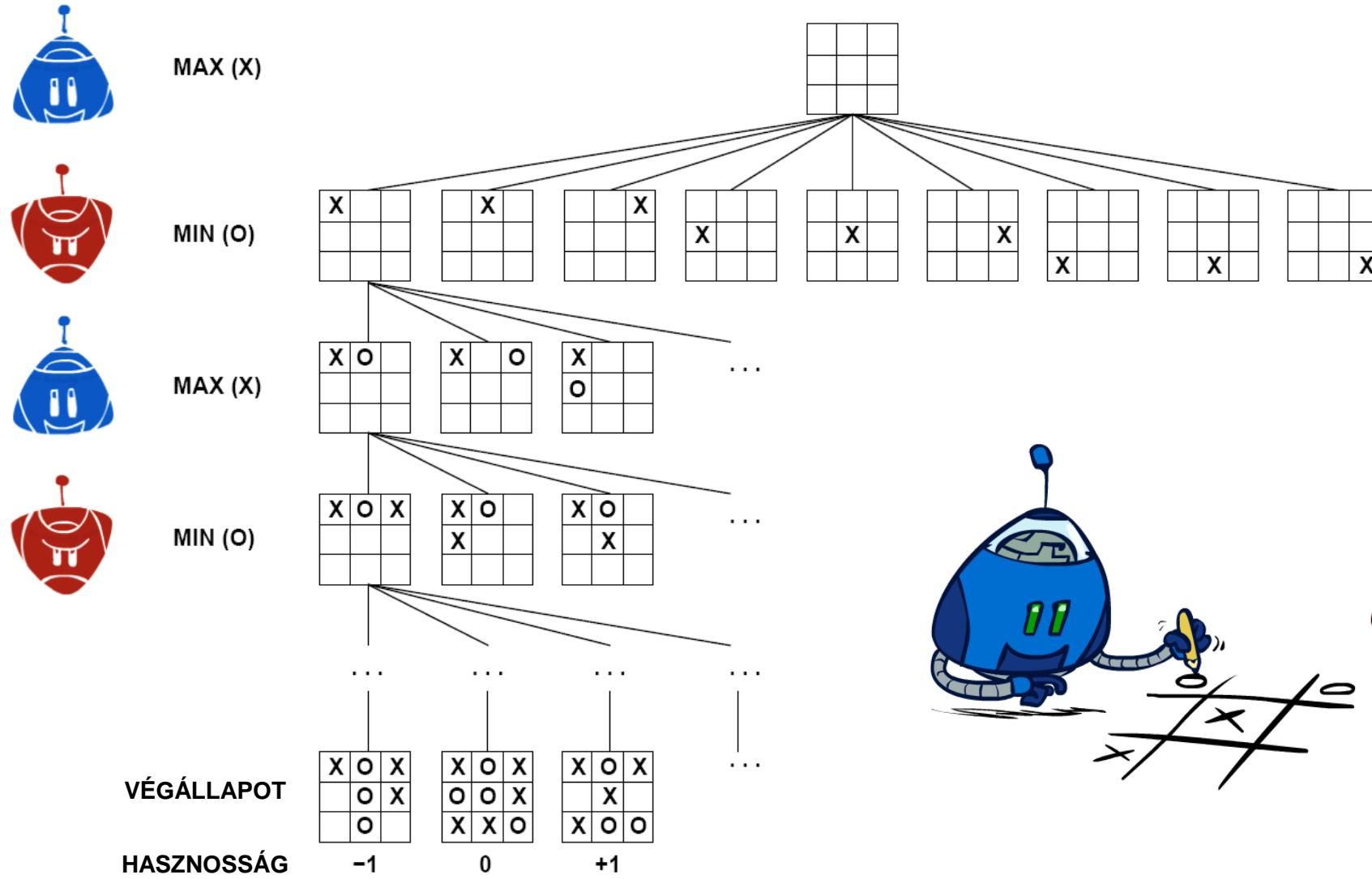


Az ellenfél által kontrollált állapotok:

$$V(s') = \min_{s \in \text{követő-állapot}(s')} V(s)$$

Végállapotok:
 $V(s) = \text{ismert}$

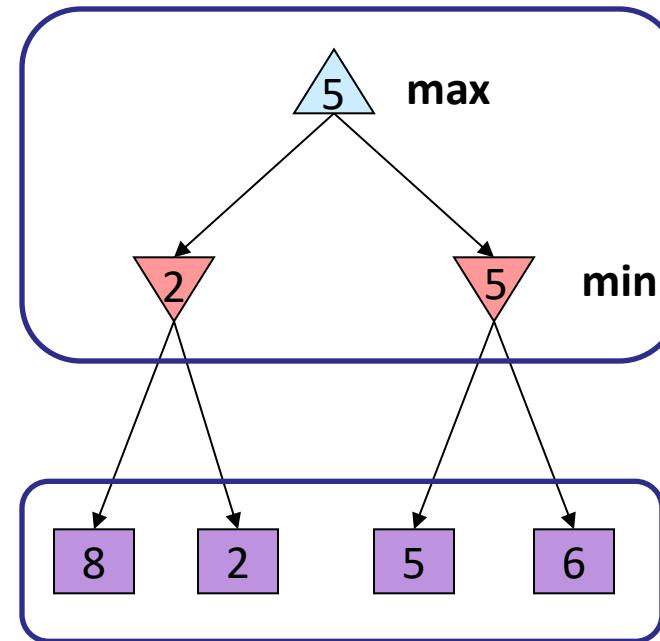
Tic-Tac-Toe (3×3-as amőba) játékfa



Keresés ellenség mellett (Minimax)

- Determinisztikus, zéró összegű játékok:
 - Tic-tac-toe, sakk, dáma stb.
 - Az egyik játékos maximalizálja az eredményt
 - A másik játékos minimalizálja az eredményt
- Minimax keresés:
 - Keresés állapottér fában (játékfa)
 - A játékosok felváltva lépnek
 - Kiszámítja minden csomópont **minimax értékét**: a legnagyobb elérhető hasznosság egy racionális (optimálisan játszó) ellenséggel szemben

Minimax értékek:
rekurzívan kiszámítva



Végállapotok értéke:
a játék szabályaiból kiszámítva

Minimax implementáció

```
def max-érték(állapot):  
    inicializál v = -∞  
    for each követő-állapot of állapot:  
        v = max(v, min-érték(követő-állapot))  
    return v
```



```
def min-érték(állapot):  
    inicializál v = +∞  
    for each követő-állapot of állapot:  
        v = min(v, max-érték(követő-állapot))  
    return v
```

$$V(s) = \max_{s' \in \text{követő-állapot}(s)} V(s')$$

$$V(s') = \min_{s \in \text{követő-állapot}(s')} V(s)$$

Minimax implementáció

```
def érték(állapot):
```

if állapot egy végállapot: **return** az állapot hasznossága
 if a következő ágens MAX: **return** max-érték(állapot)
 if a következő ágens MIN: **return** min-érték(állapot)

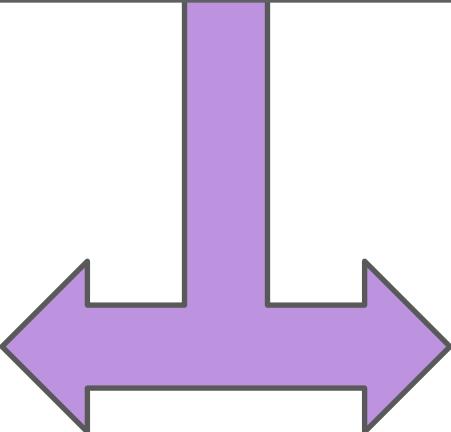
```
def max-érték(állapot):
```

inizializál v = -∞

for each követő-állapot **of** állapot:

 v = max(v, érték(követő-állapot))

return v



```
def min-érték(állapot):
```

inizializál v = +∞

for each követő-állapot **of** állapot:

 v = min(v, érték(követő-állapot))

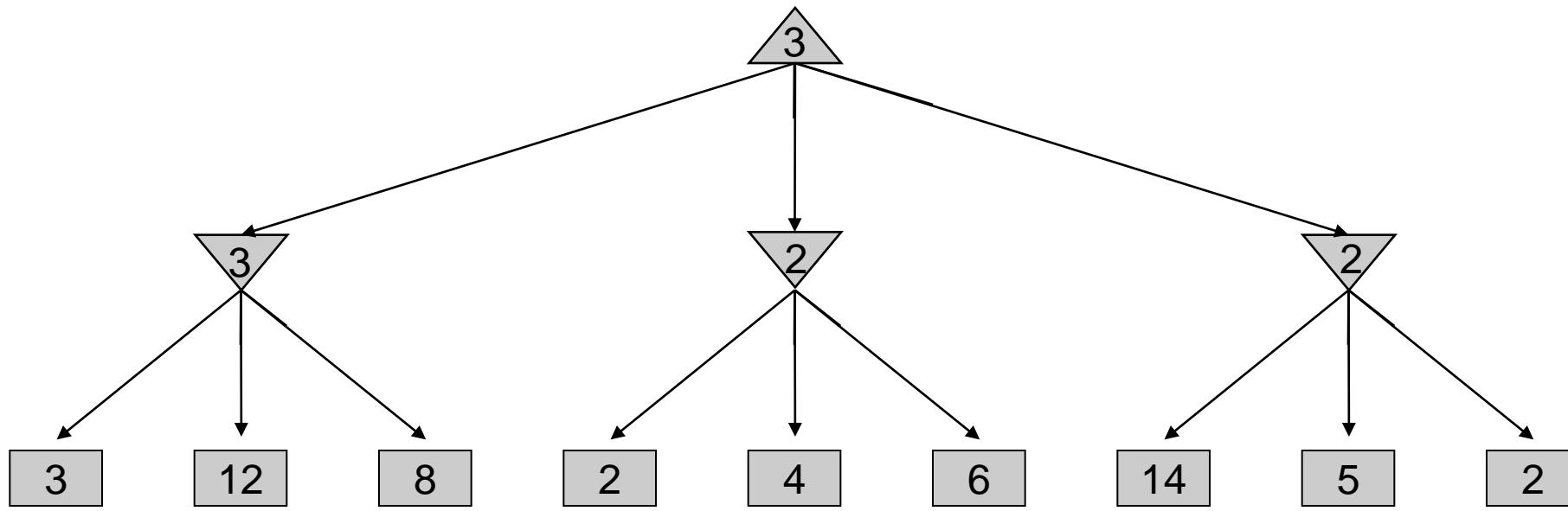
return v

Minimax példa

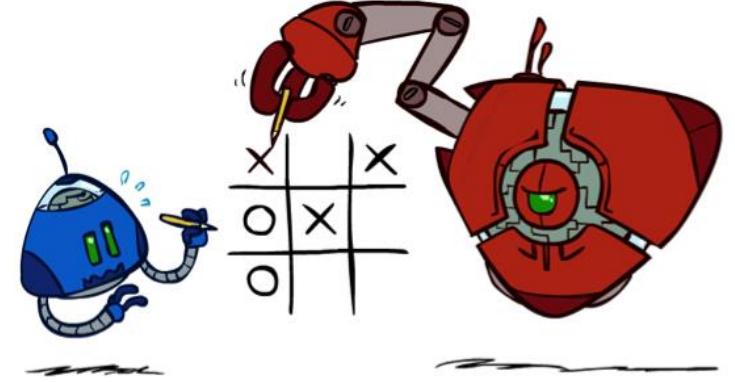
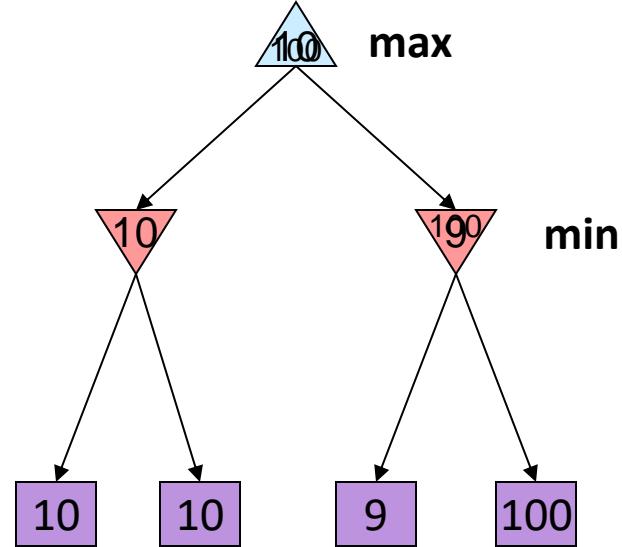
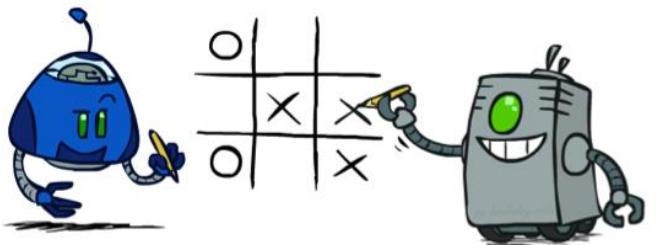
MAX

MIN

VÉG-
ÁLLAPOT



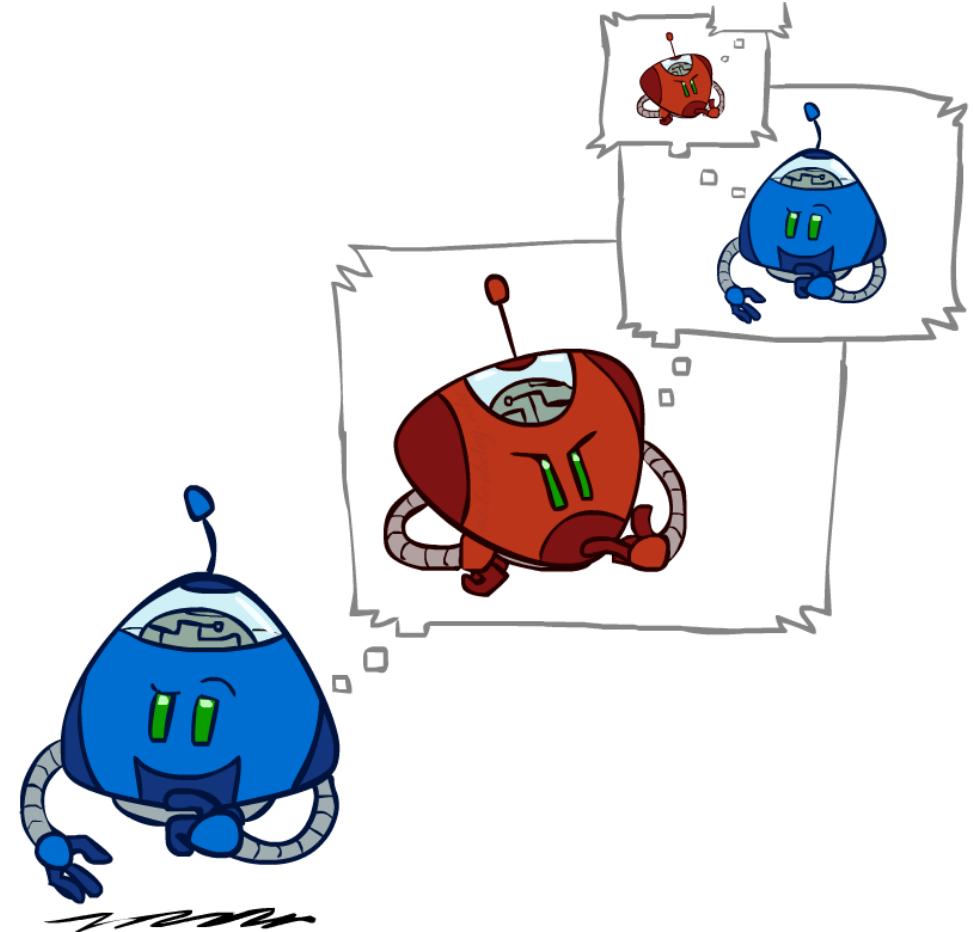
Minimax tulajdonságok



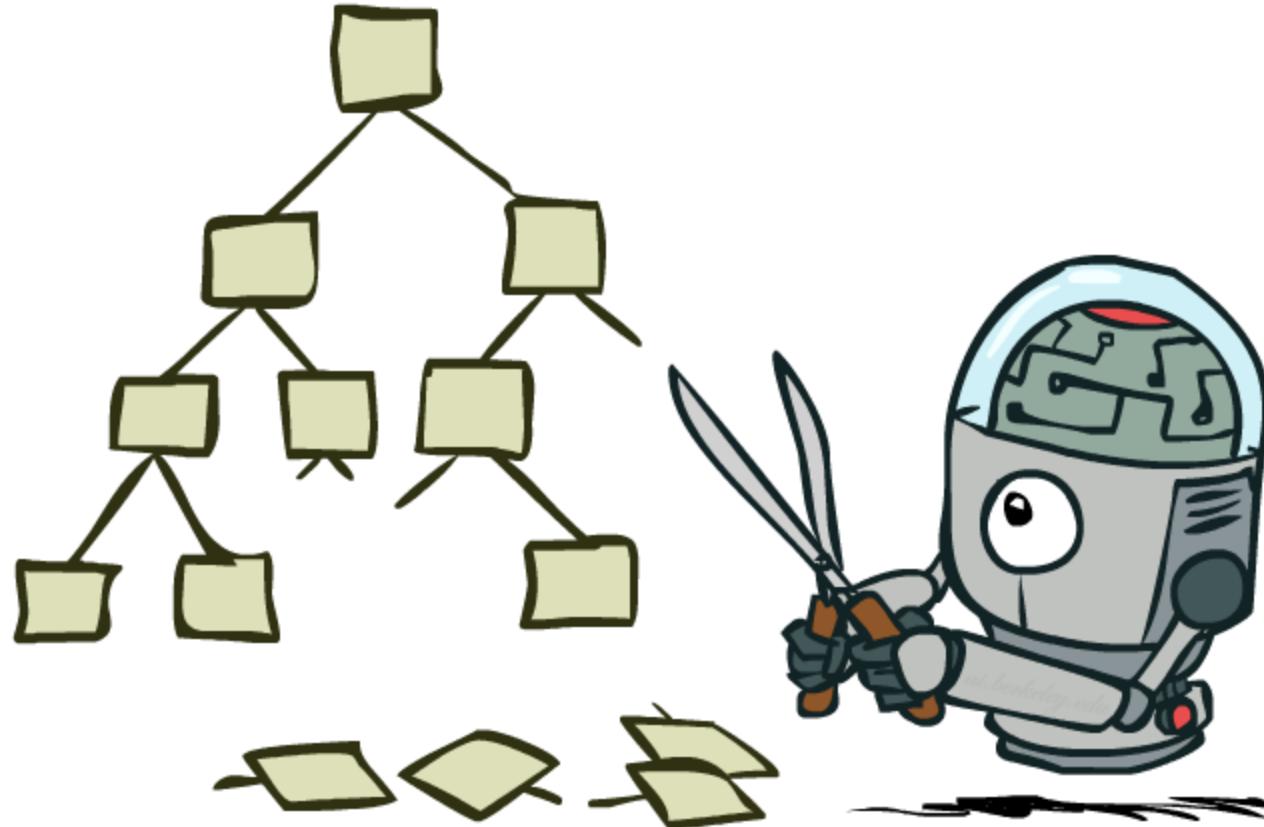
Optimális egy racionális játékossal szemben.
Egyébként?

Minimax hatékonysága

- Milyen hatékony a minimax algoritmus?
 - Ugyanannyira, mint a (kimerítő) DFS
 - Idő: $O(b^m)$
 - Tár: $O(bm)$
- Példa: Sakk, $b \approx 35$, $m \approx 100$
 - Egzakt megoldás kivitelezhetetlen
 - Szükséges-e felderítenünk a teljes játékfát?



A játékfa nyesése

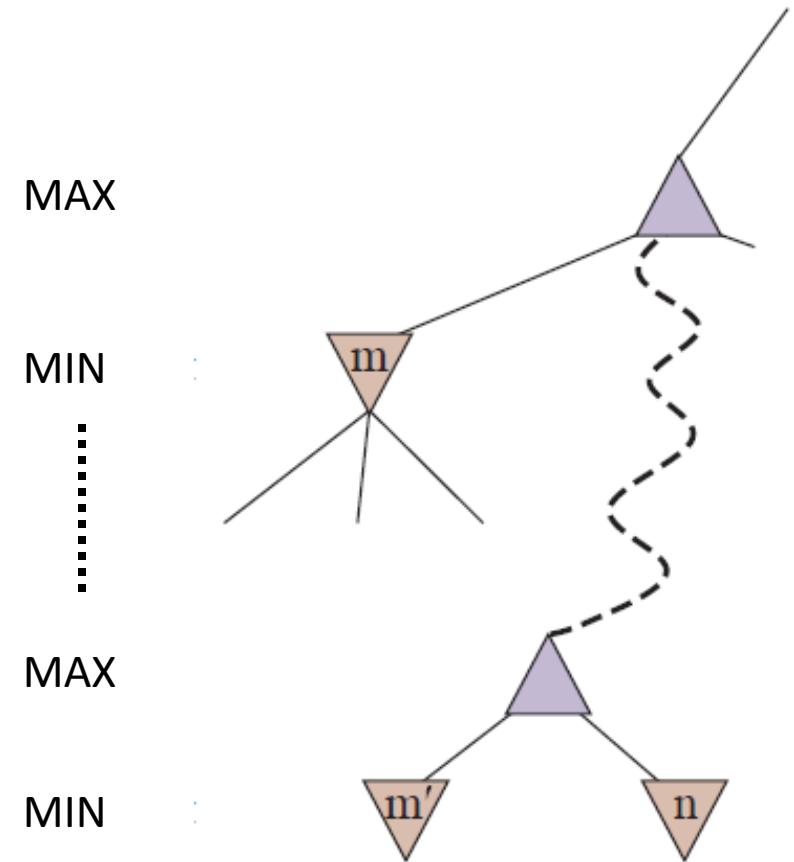


Alfa-béta nyesés

■ Általános eset (MIN szerint)

- Éppen az n csomópont MIN-ÉRTÉKét számítjuk ki
- Végiglépkedünk n gyerekein
- n gyerekeinek minimum értéke egyre csökken(het)
- Kinek fontos n értéke? MAX-nak
- Legyen m és m' két másik, már megvizsgált csomópont, amelyeket MAX választhat
- Ha n rosszabbá válik, mint m vagy m' , akkor MAX el fogja kerülni n -t (tudjuk, hogy rossz annyira, hogy MAX nem fogja választani), így nem kell megvizsgálnunk n további gyerekeit

■ MAX szerint szimmetrikus



Alfa-béta implementáció

α : MAX legjobb választási lehetősége eddig
 β : MIN legjobb választási lehetősége eddig

def max-érték(*állapot*):

inicializál *v* = $-\infty$

for each követő-állapot **of** *állapot*:

v = max(*v*, érték(követő-állapot, α , β))

if *v* $\geq \beta$ **return** *v*

α = max(α , *v*)

return *v*

def min-érték(*állapot*):

inicializál *v* = $+\infty$

for each követő-állapot **of** *állapot*:

v = min(*v*, érték(követő-állapot, α , β))

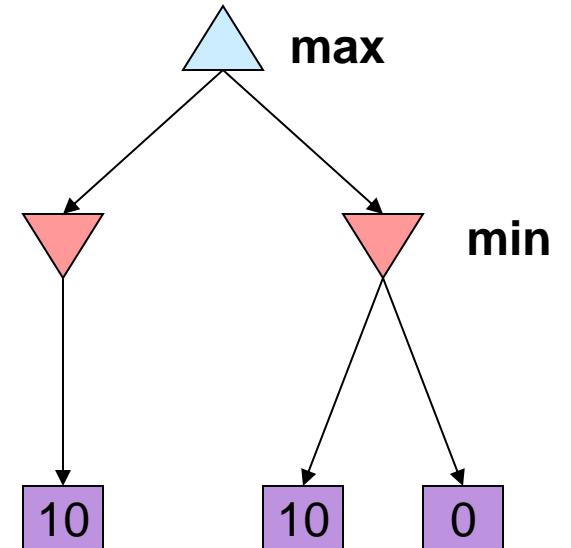
if *v* $\leq \alpha$ **return** *v*

β = min(β , *v*)

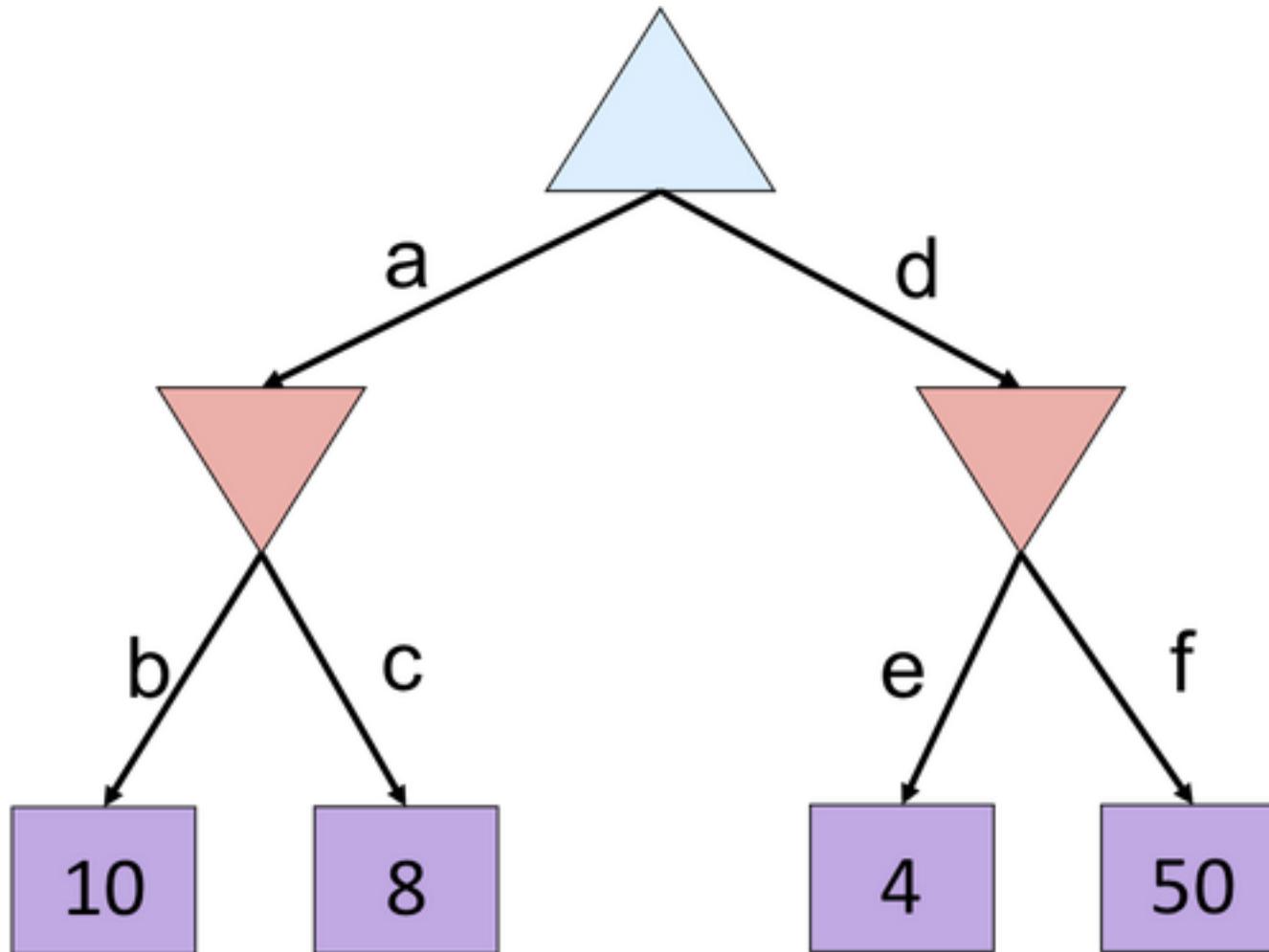
return *v*

Alfa-béta nyesés tulajdonságai

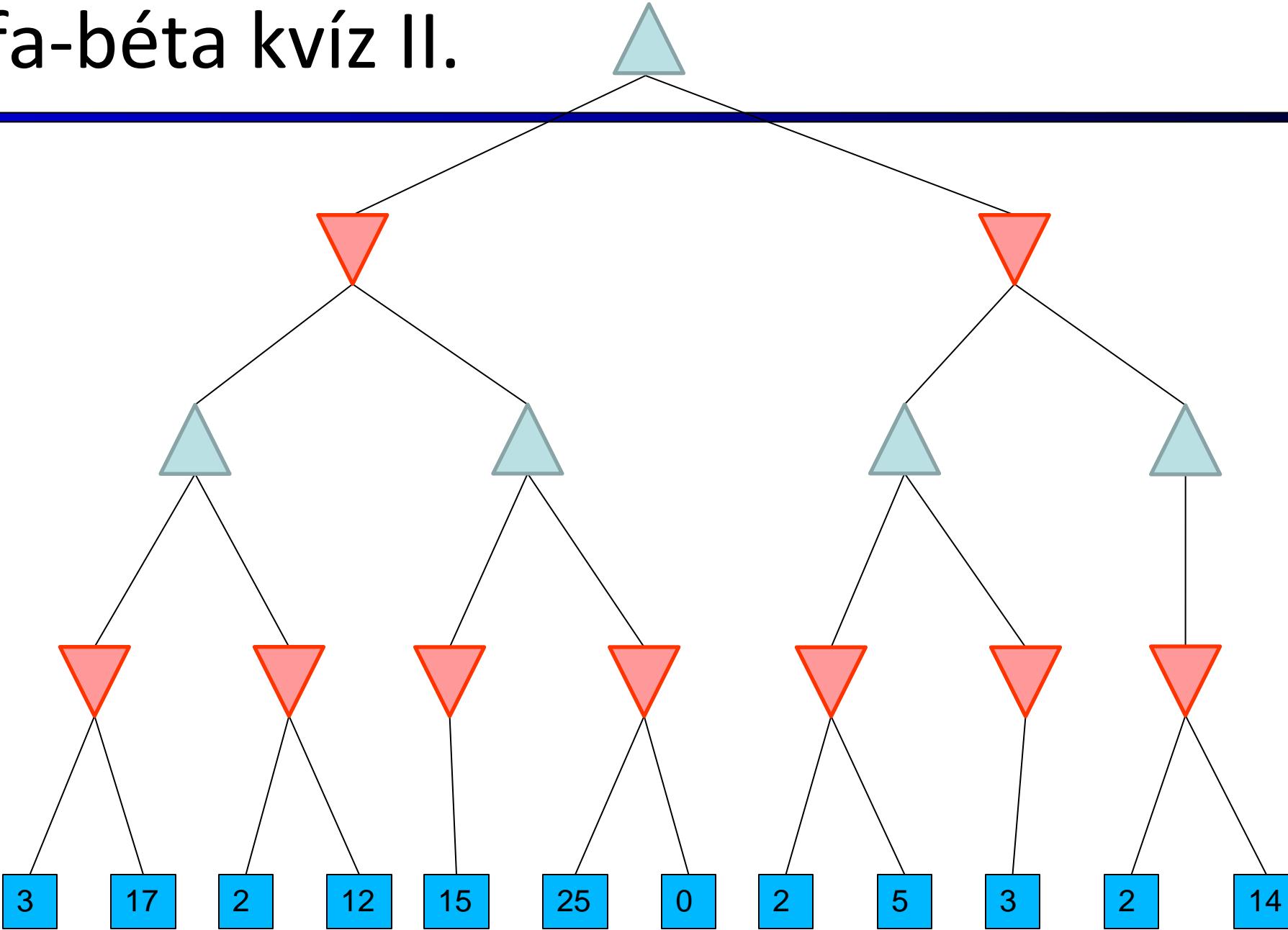
- A nyesés nem befolyásolja a gyökér csomópont minimax értékét!
- Köztes csomópontok értéke eltérhet a valóságtól
- A gyerek csomópontok jó sorrendezése növeli a nyesés hatékonyságát
- „Tökéletes” sorrendezéssel:
 - Az időkomplexitás $O(b^m)$ –ről $O(b^{m/2})$ -re csökken
= az elágazási tényező b -ről \sqrt{b} -re csökken
 - Dupla keresési mélység!
 - Sok esetben (pl. sakk) a teljes mélységű keresés továbbra sem kivitelezhető



Alfa-béta kvíz I.

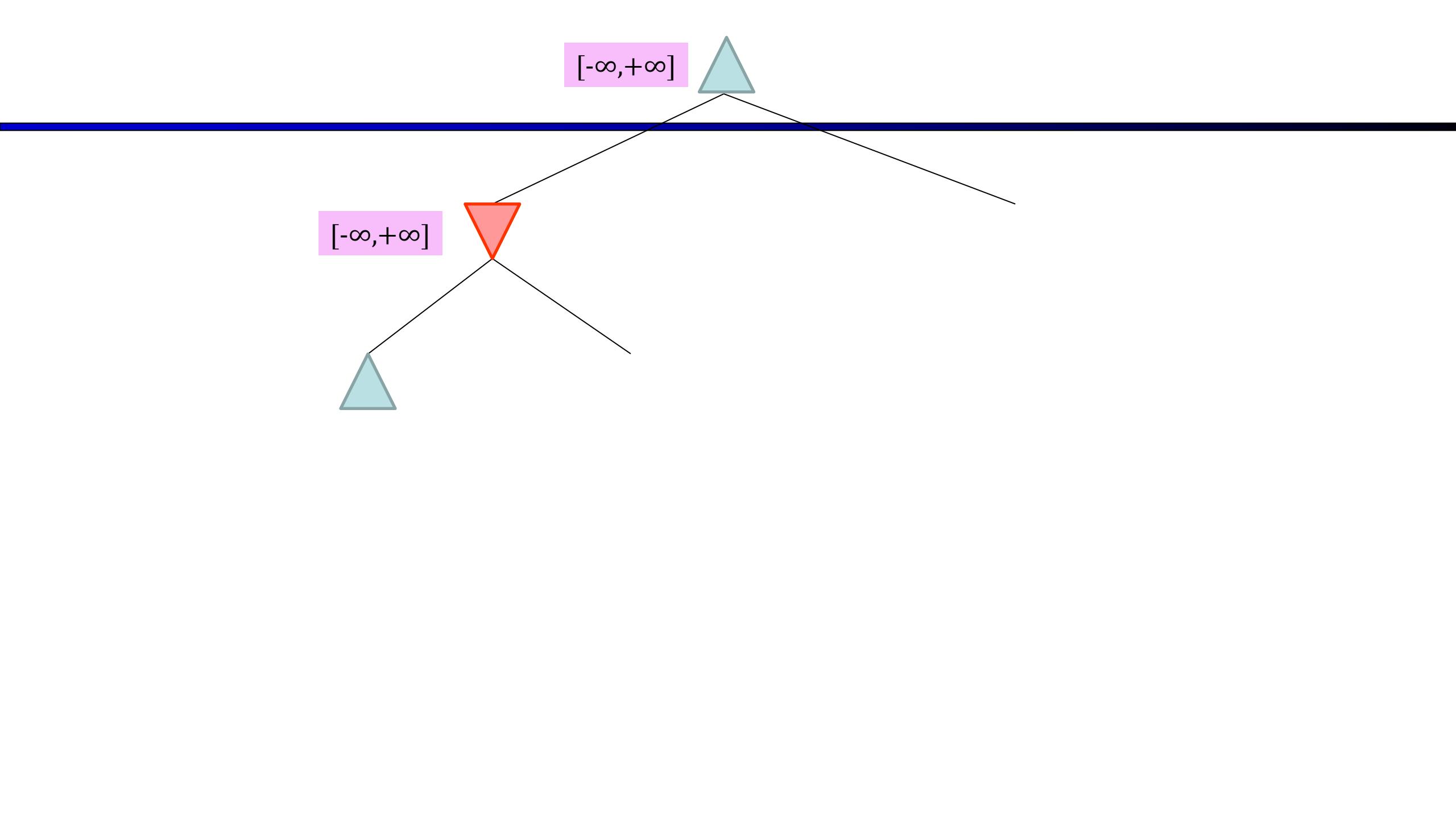


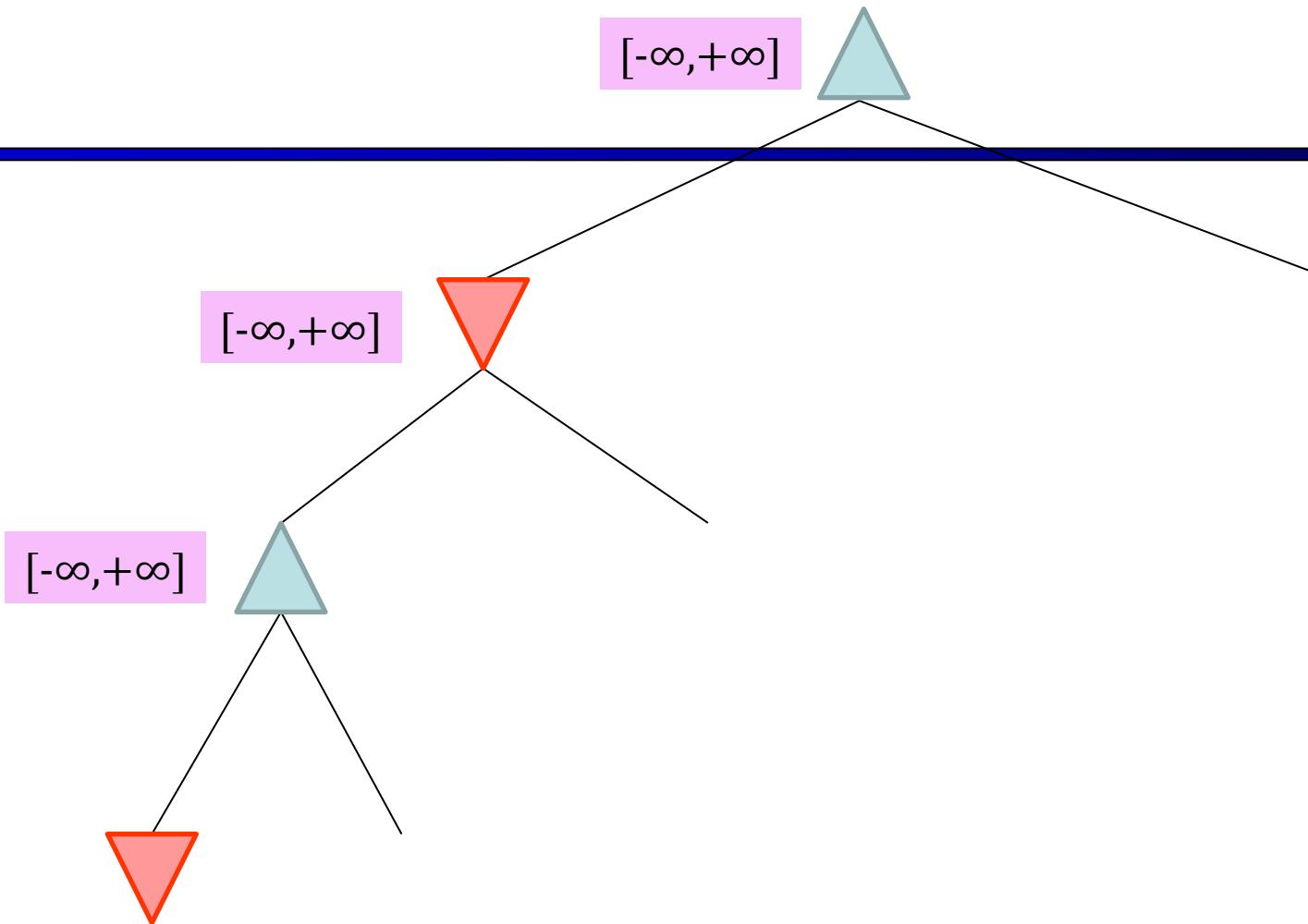
Alfa-béta kvíz II.

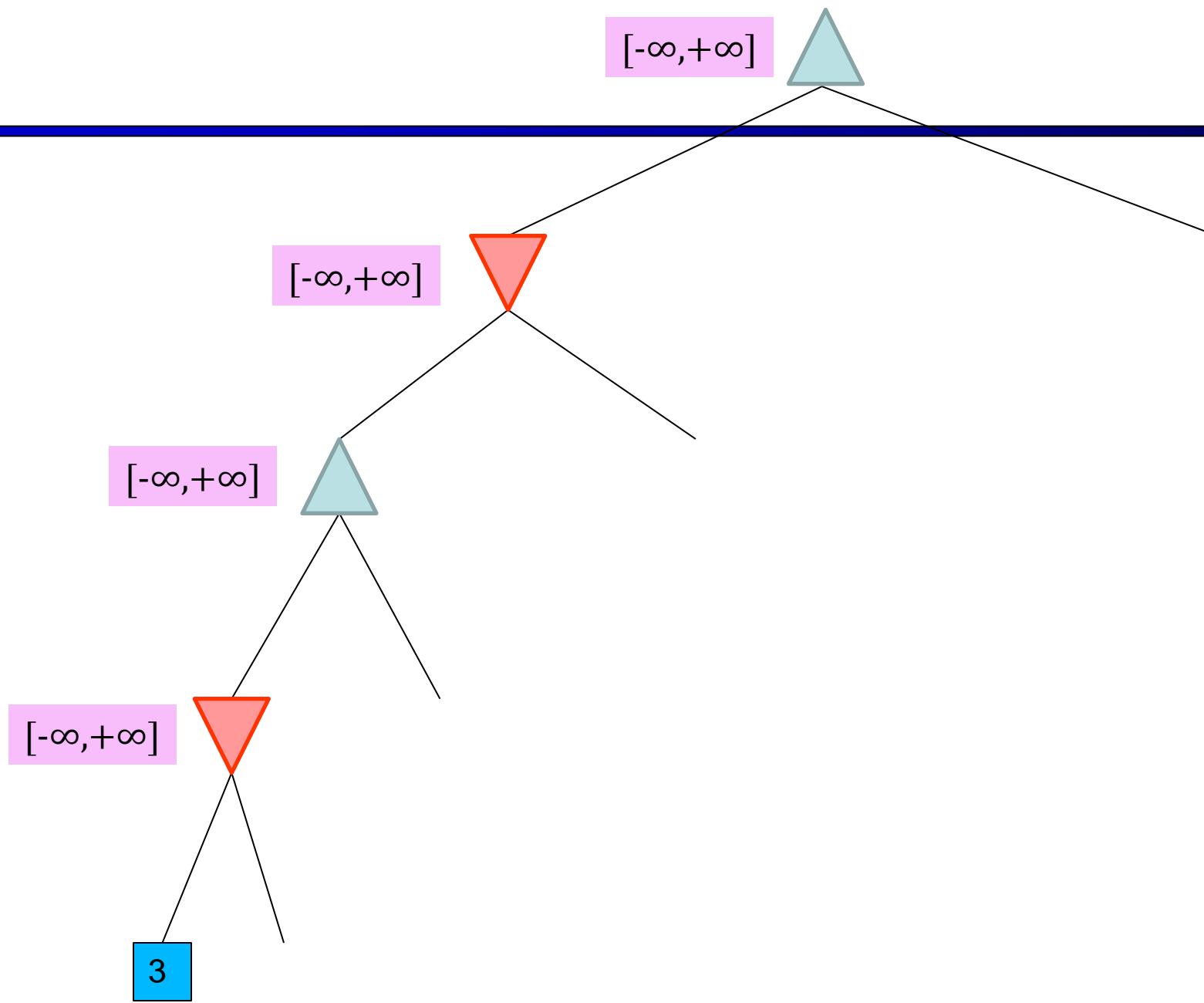


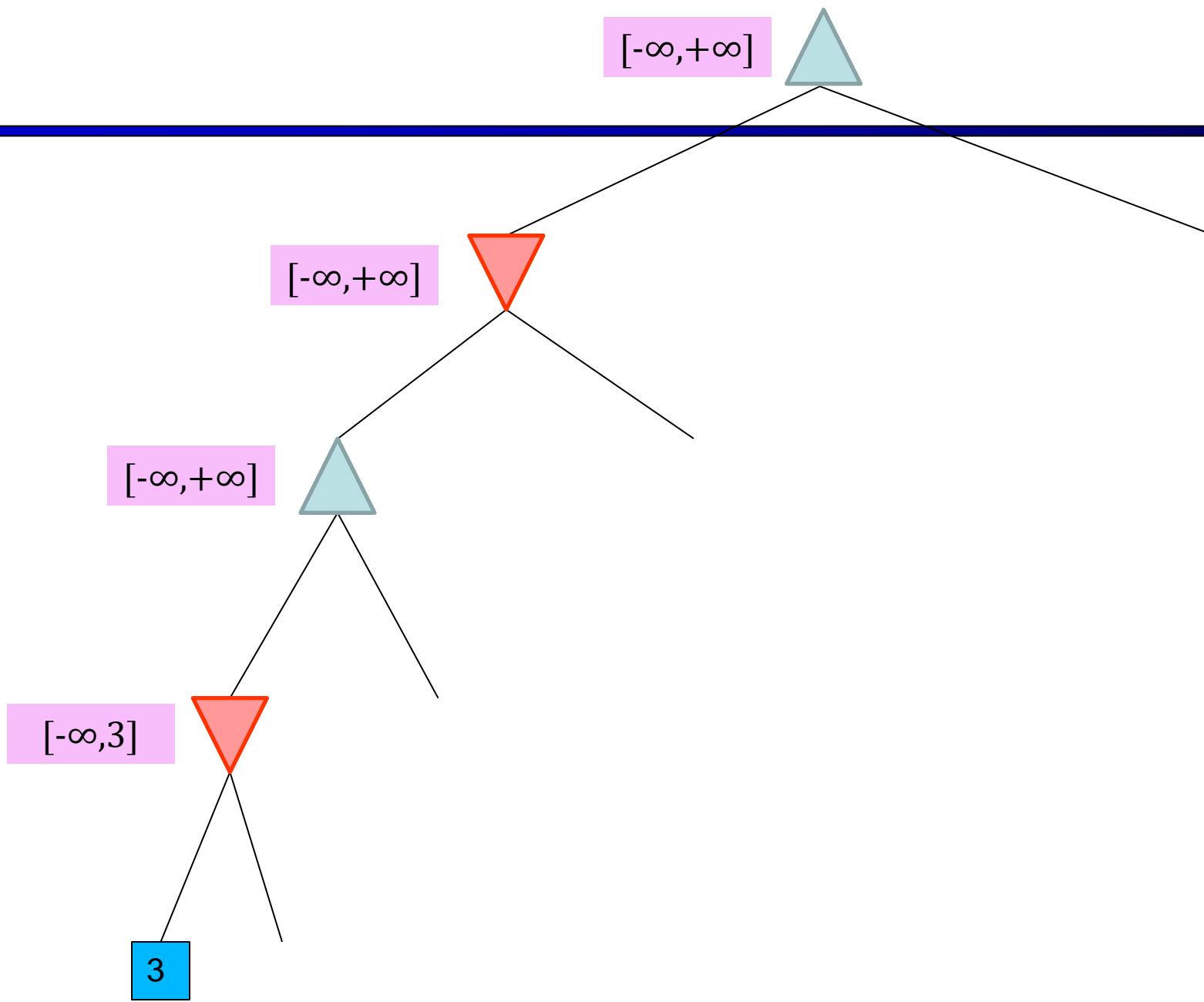
$[-\infty, +\infty]$

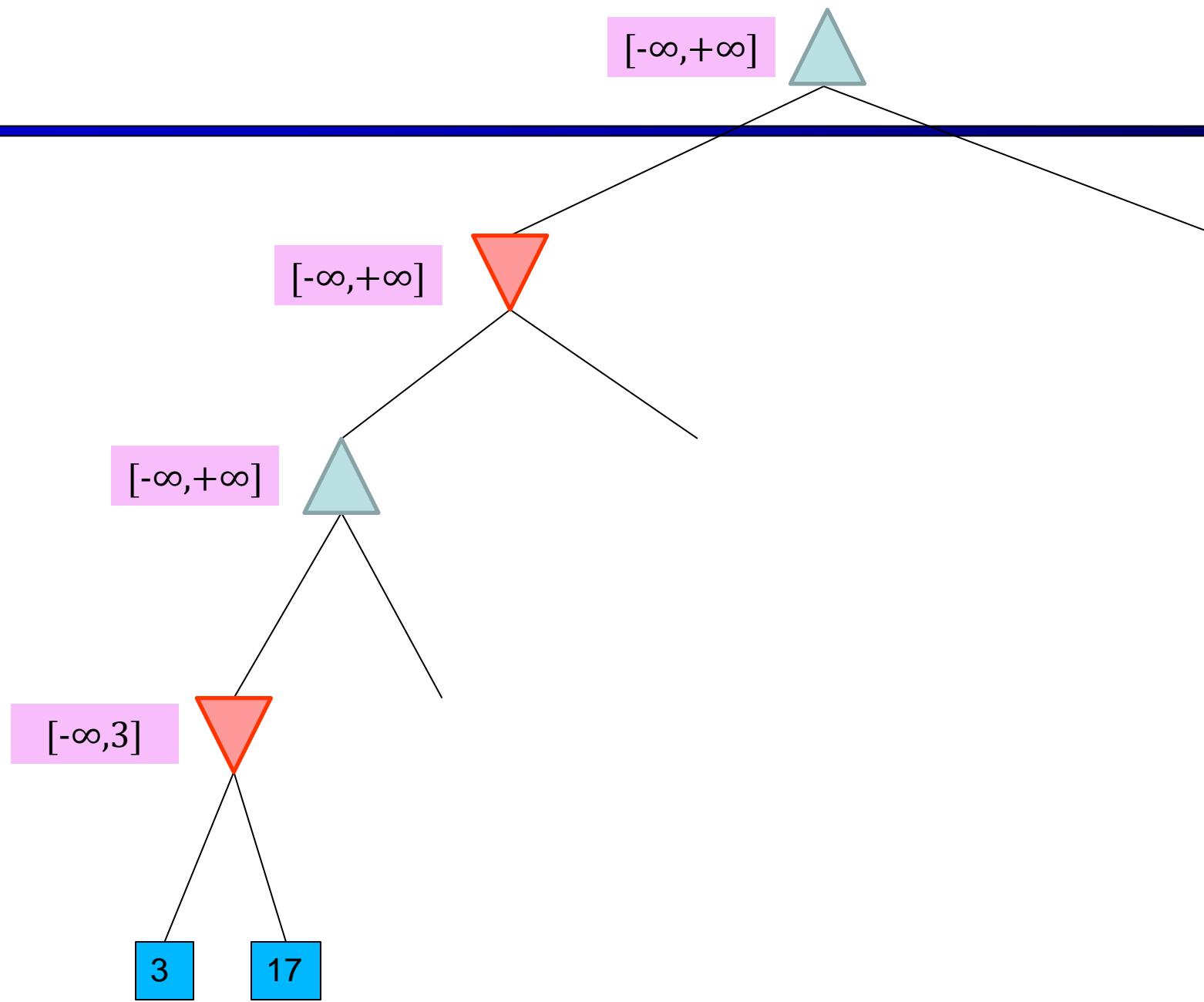


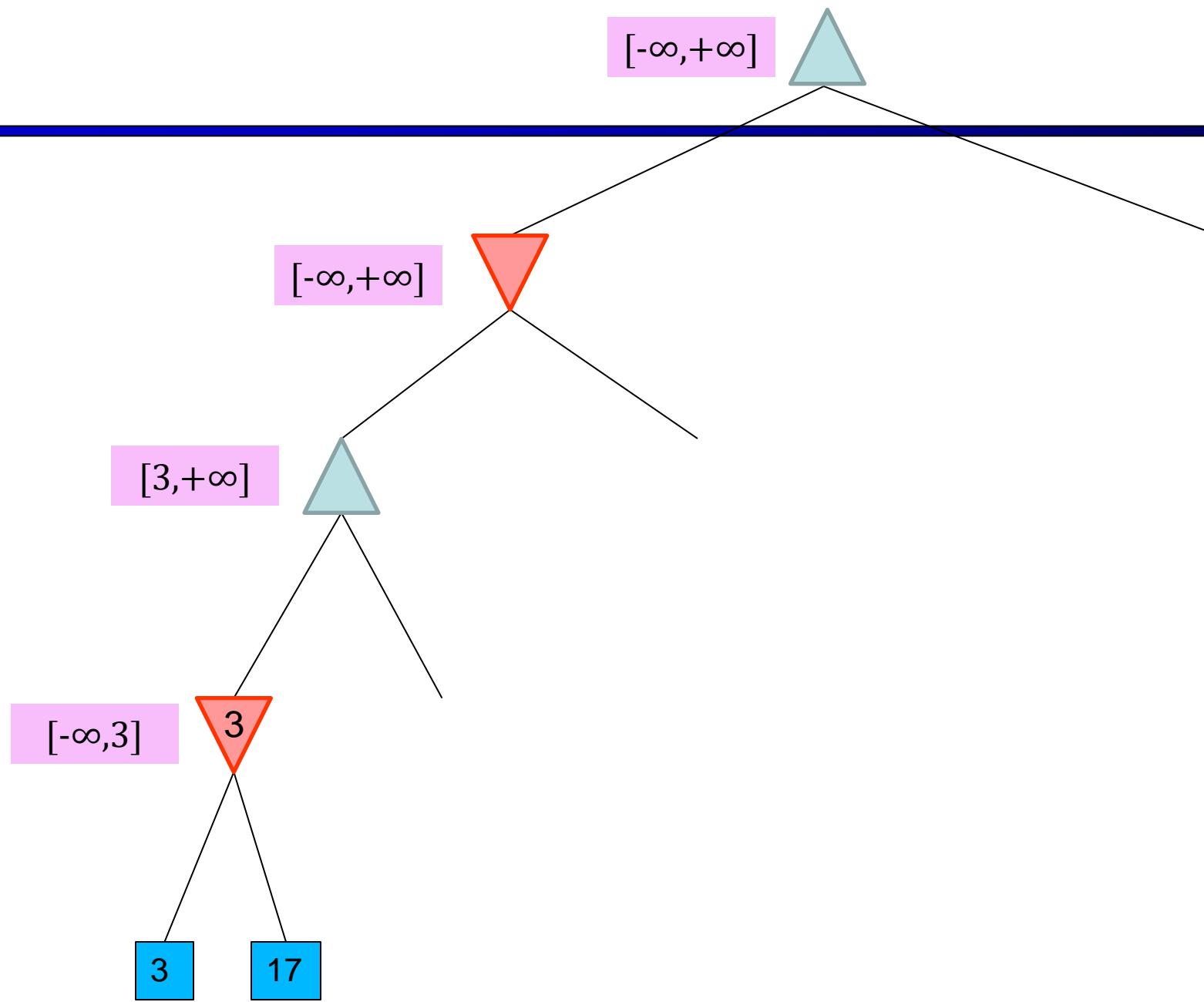


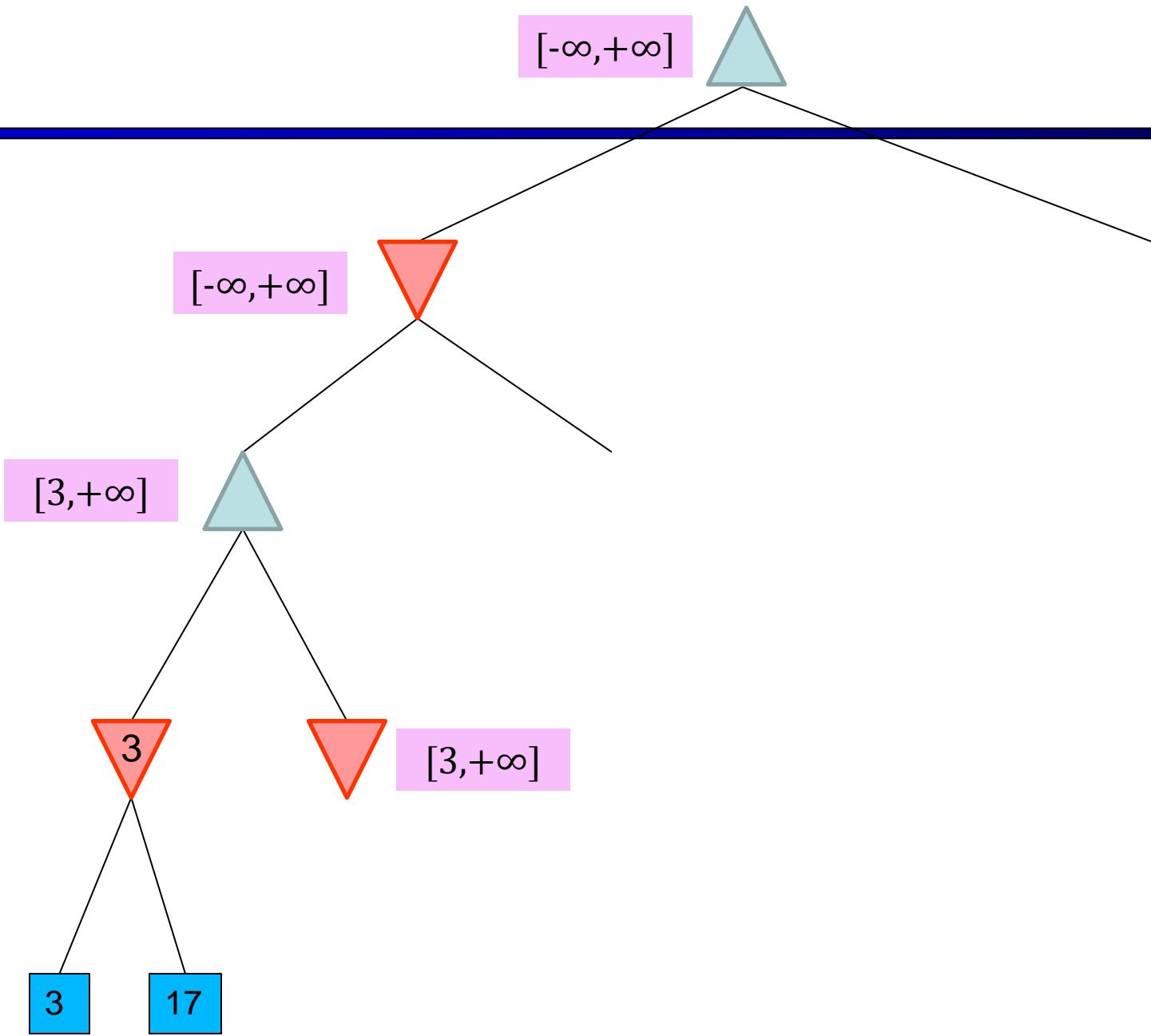


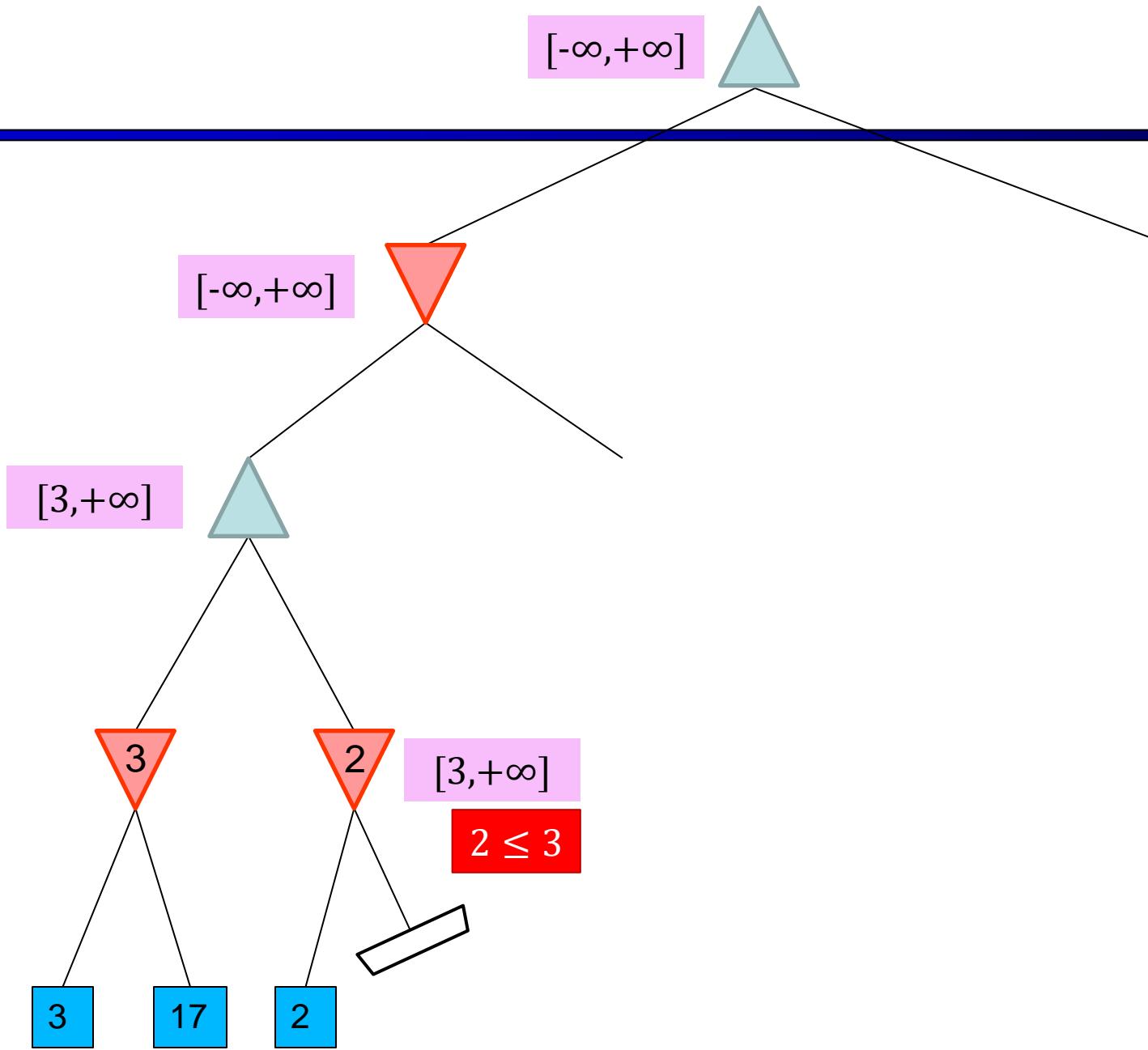


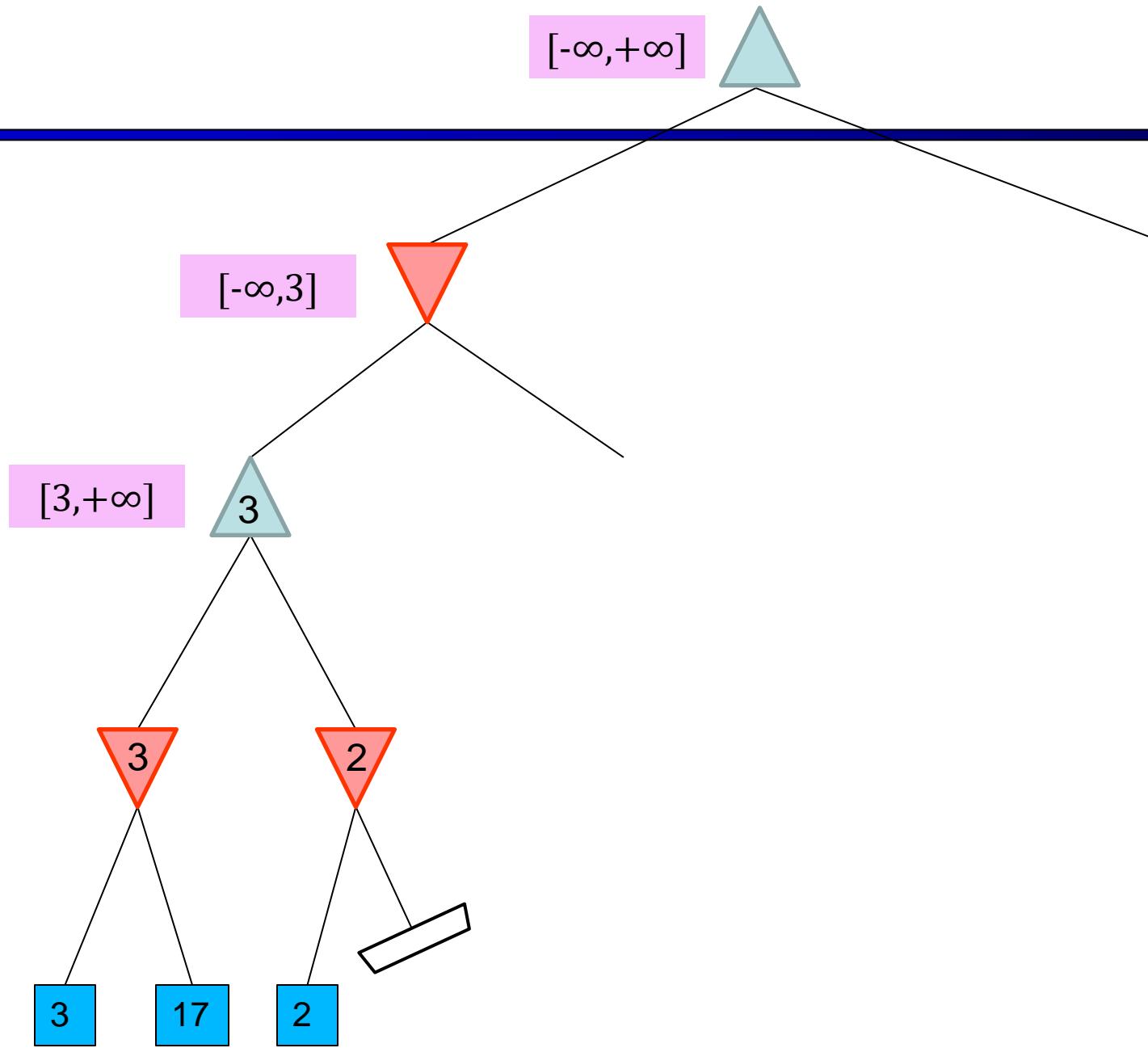


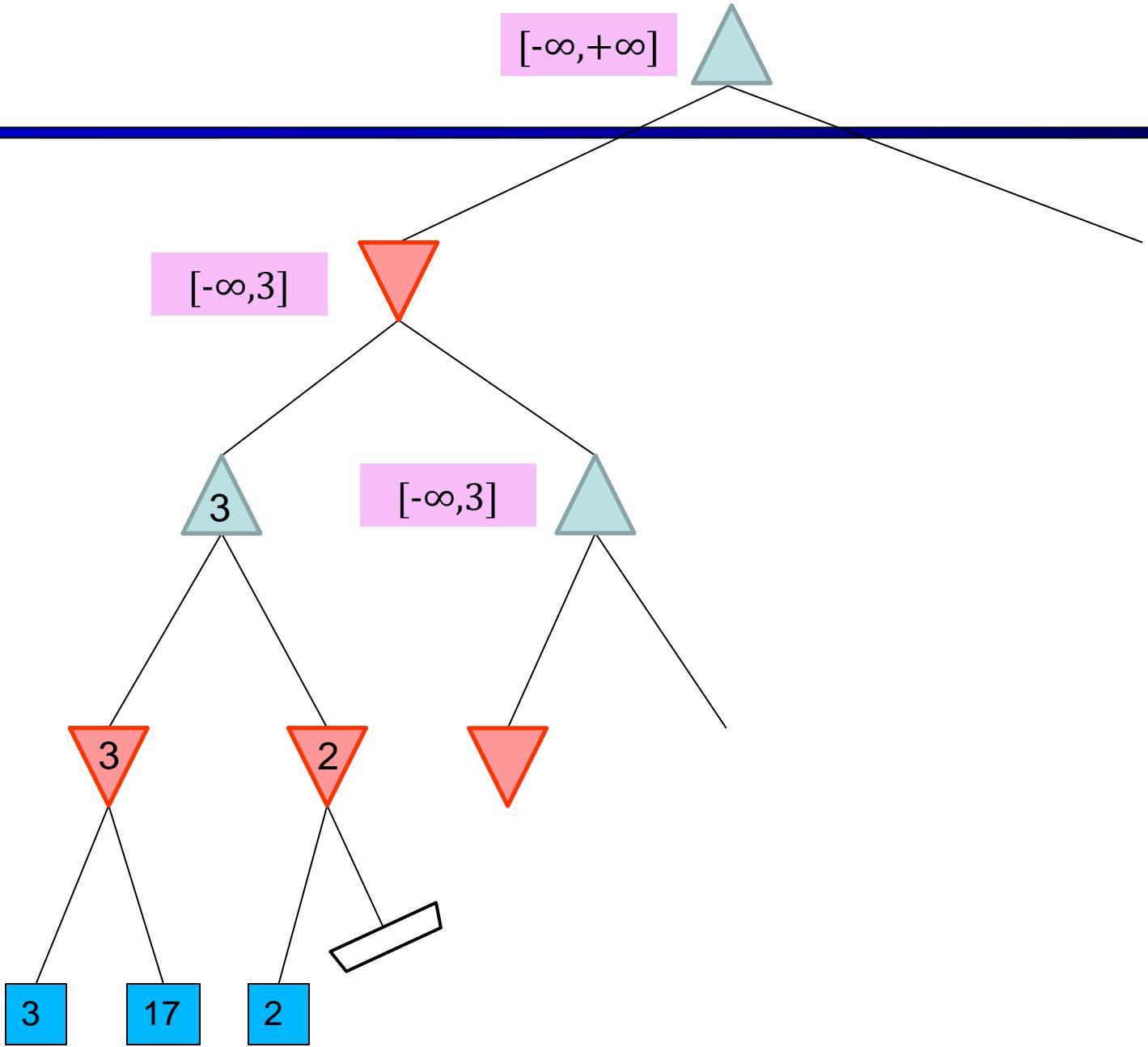


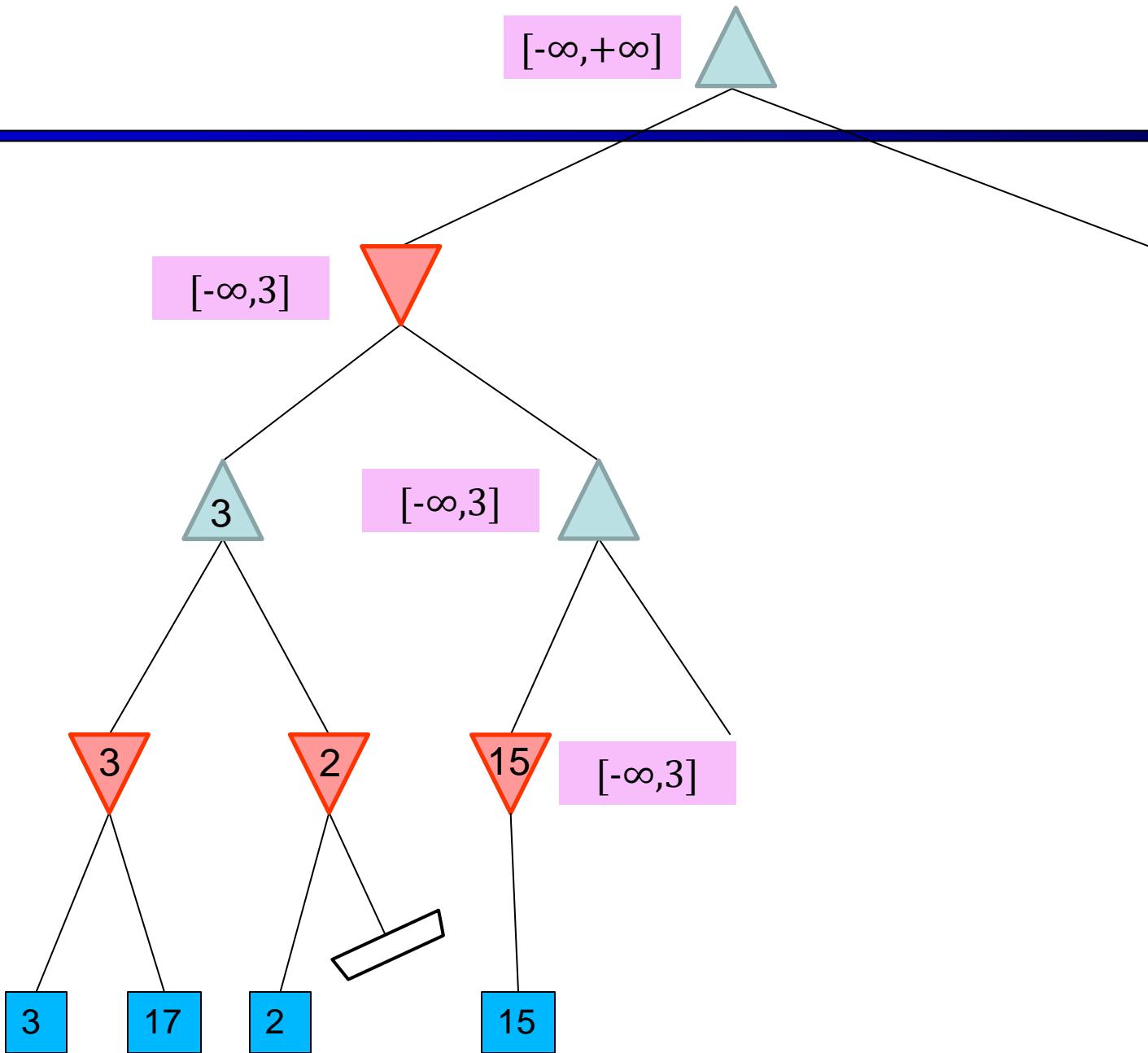


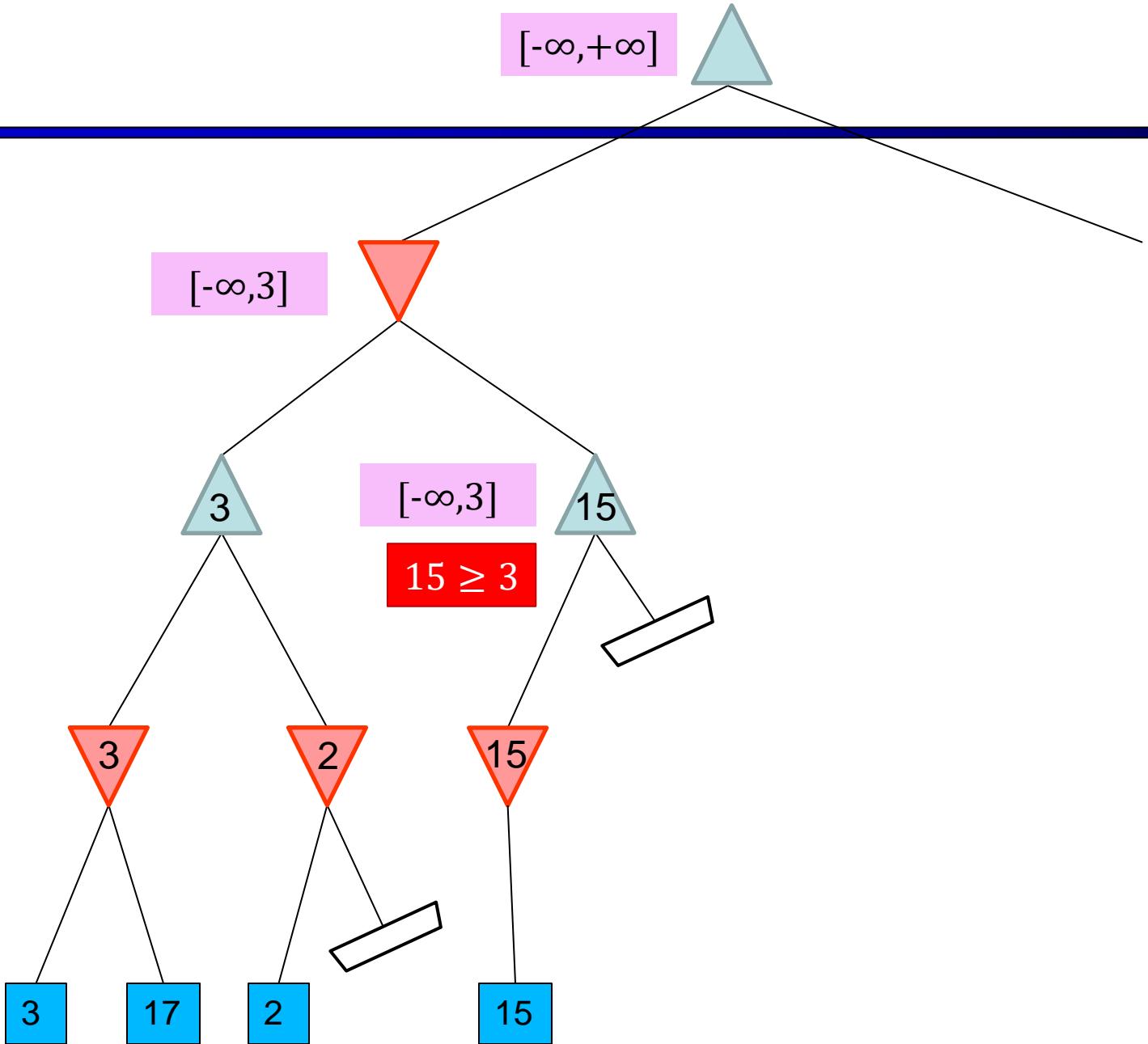


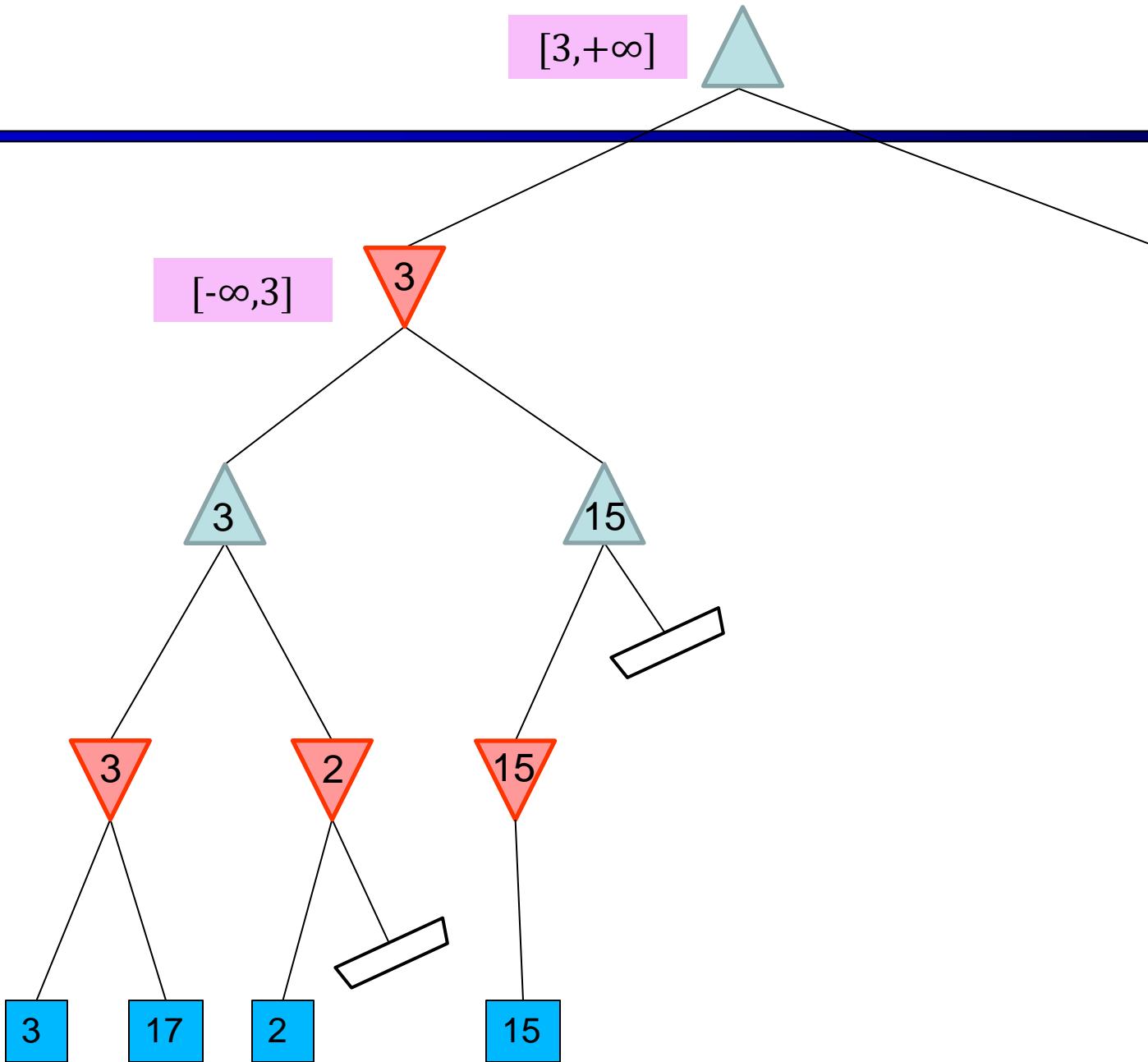


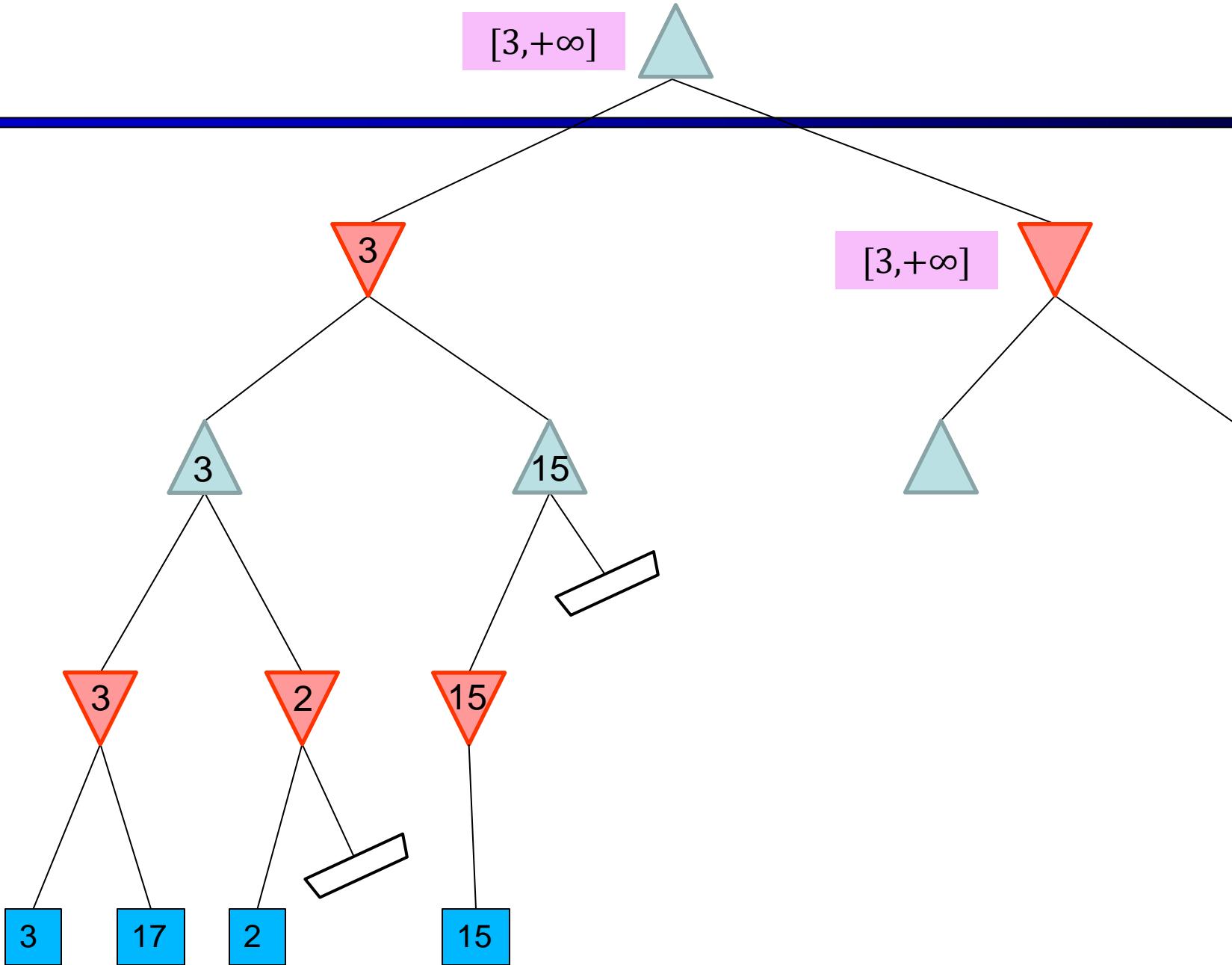


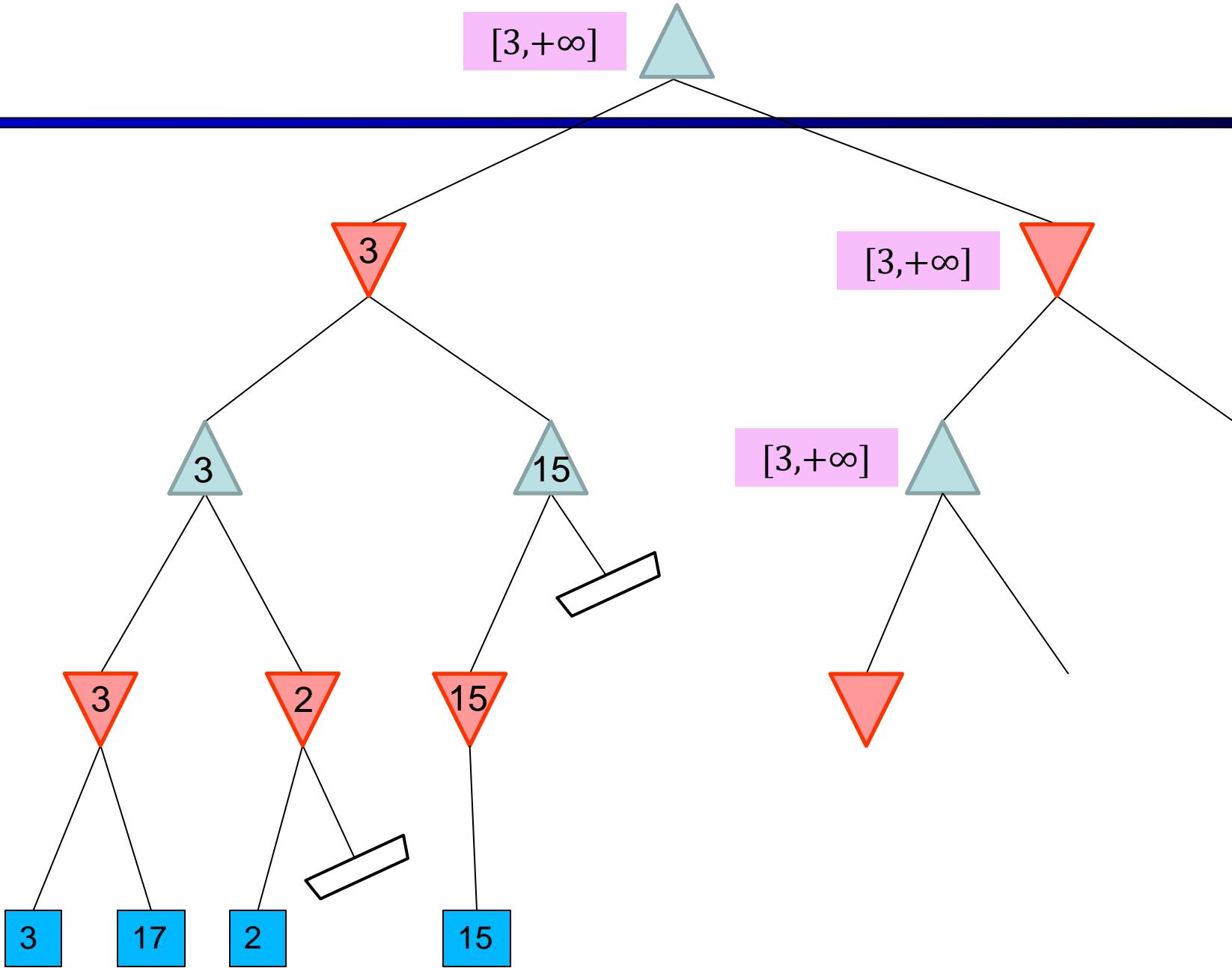


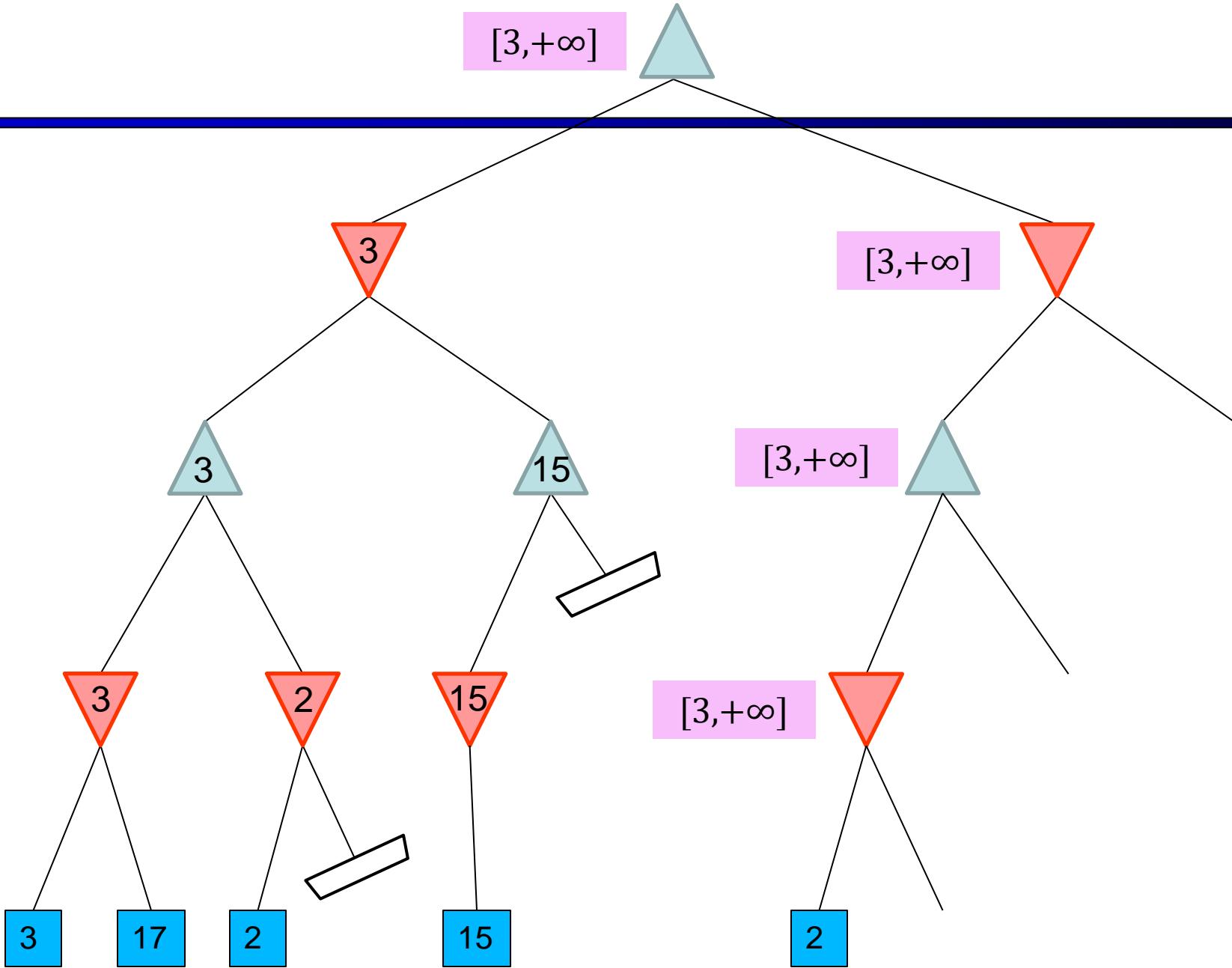


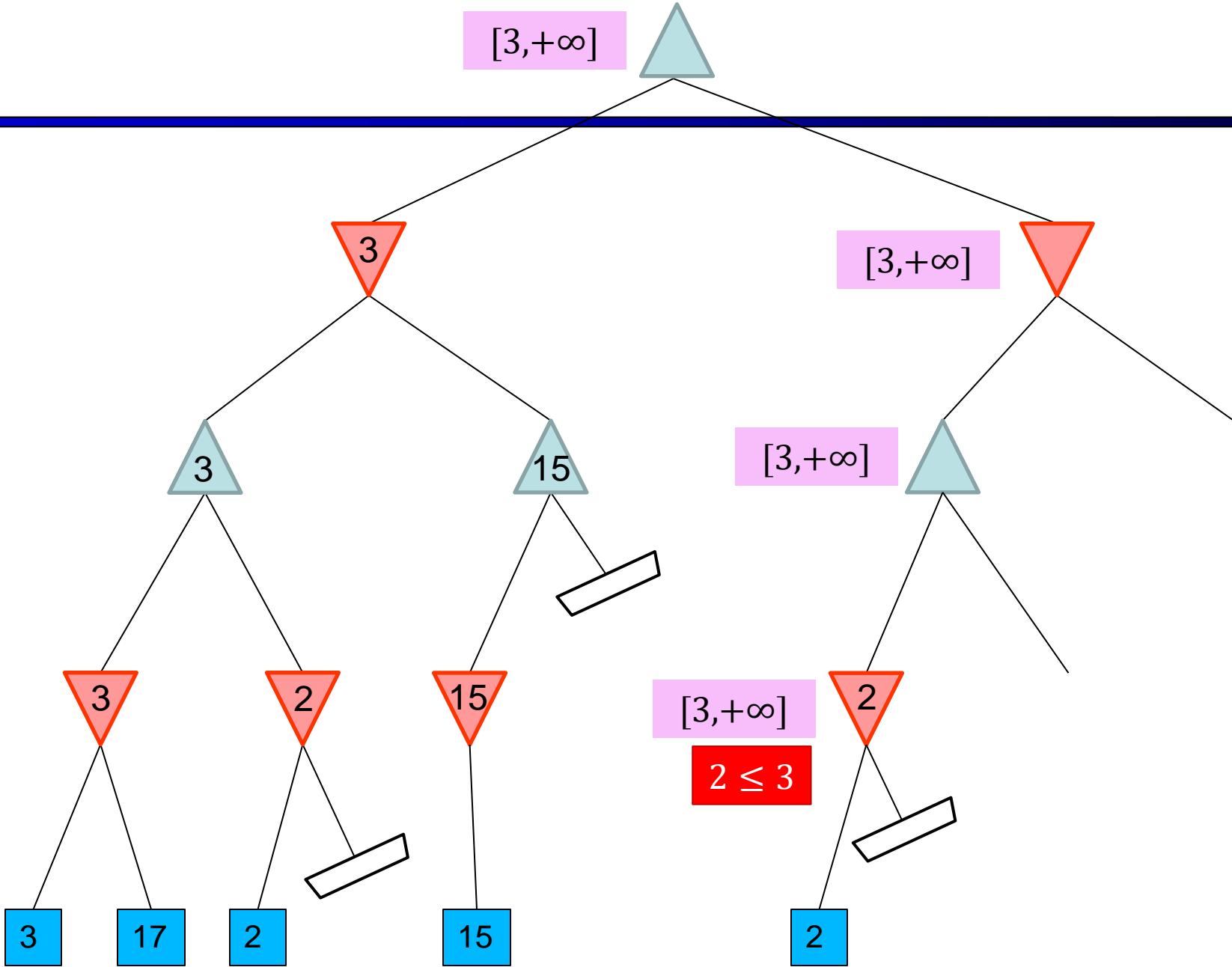


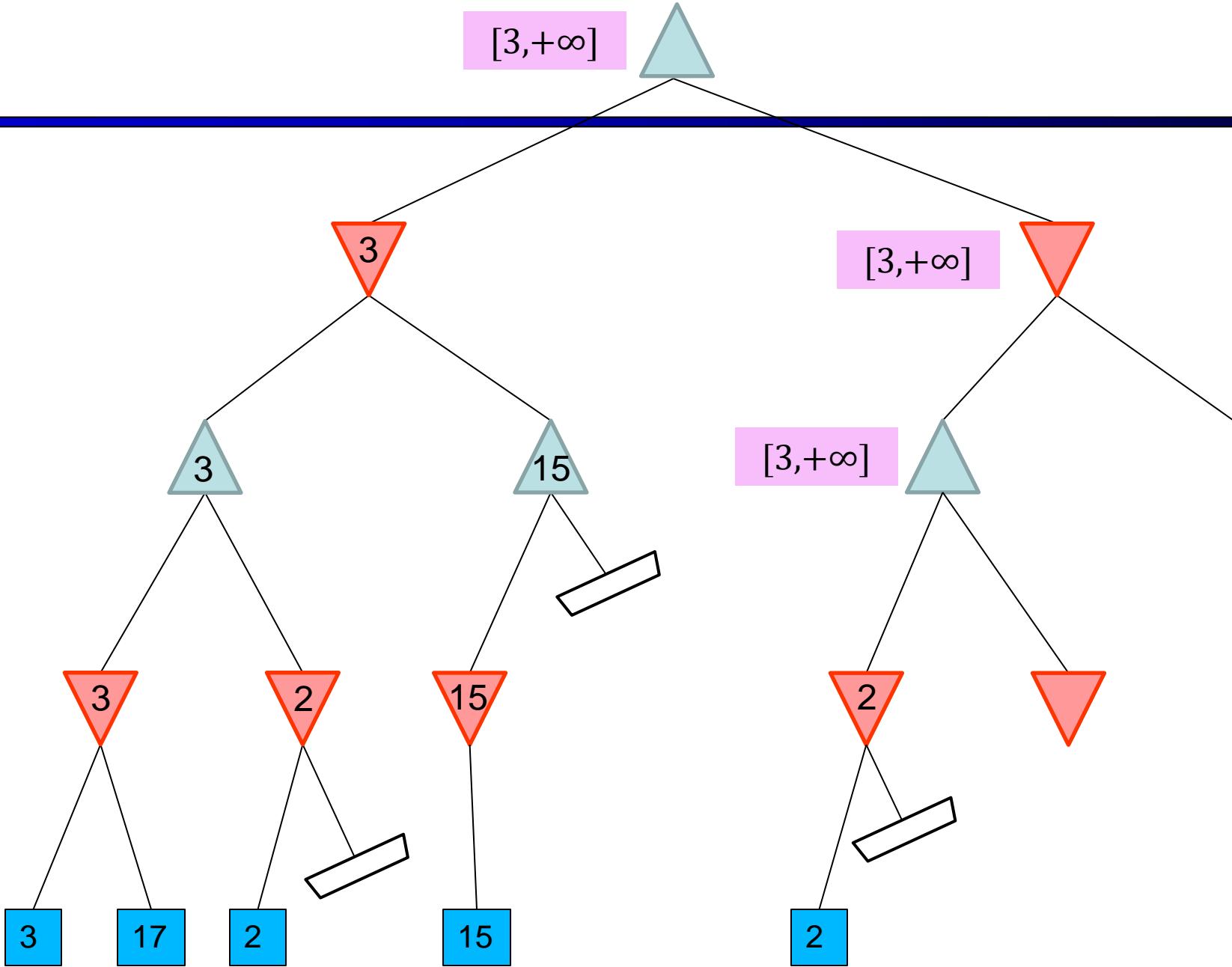


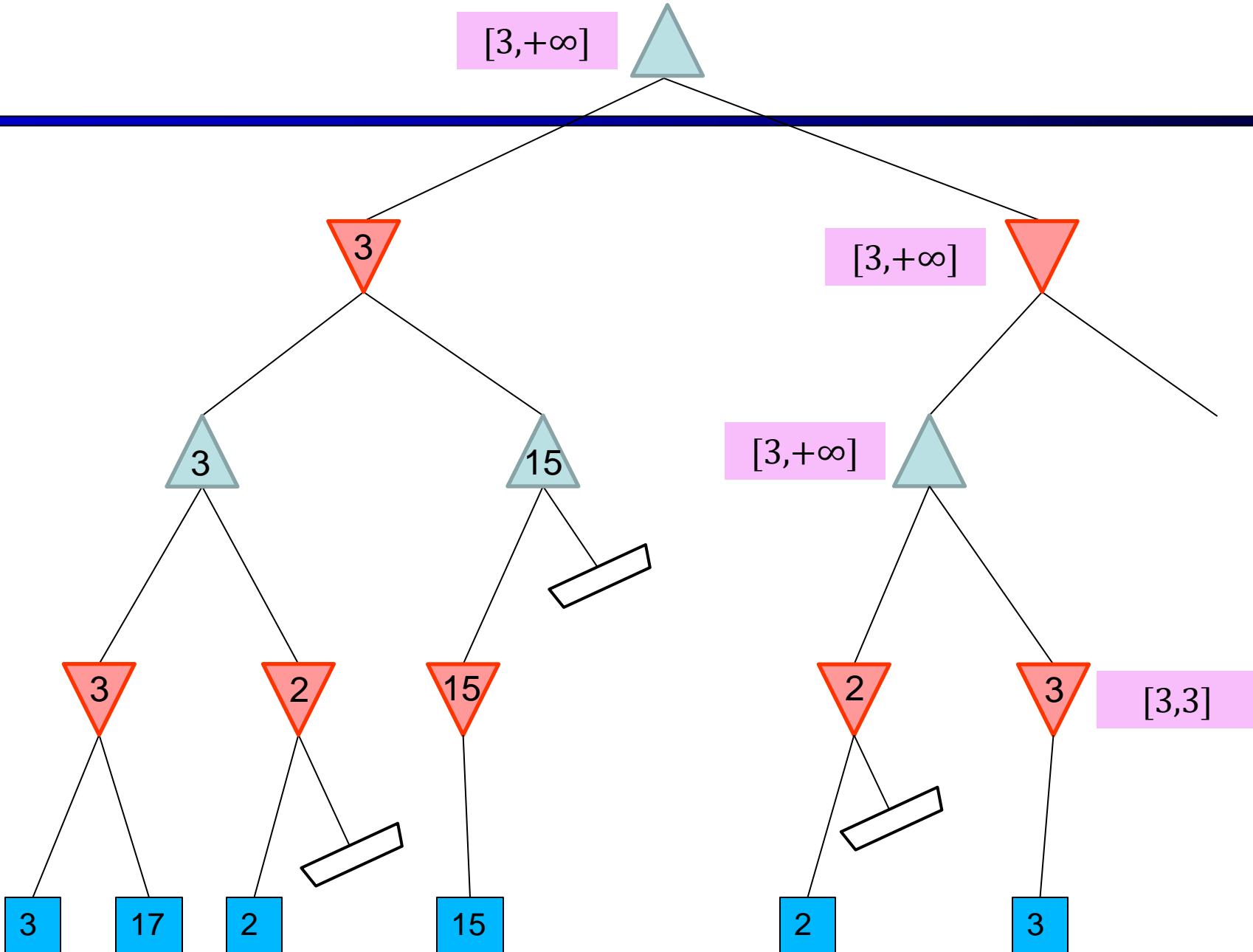


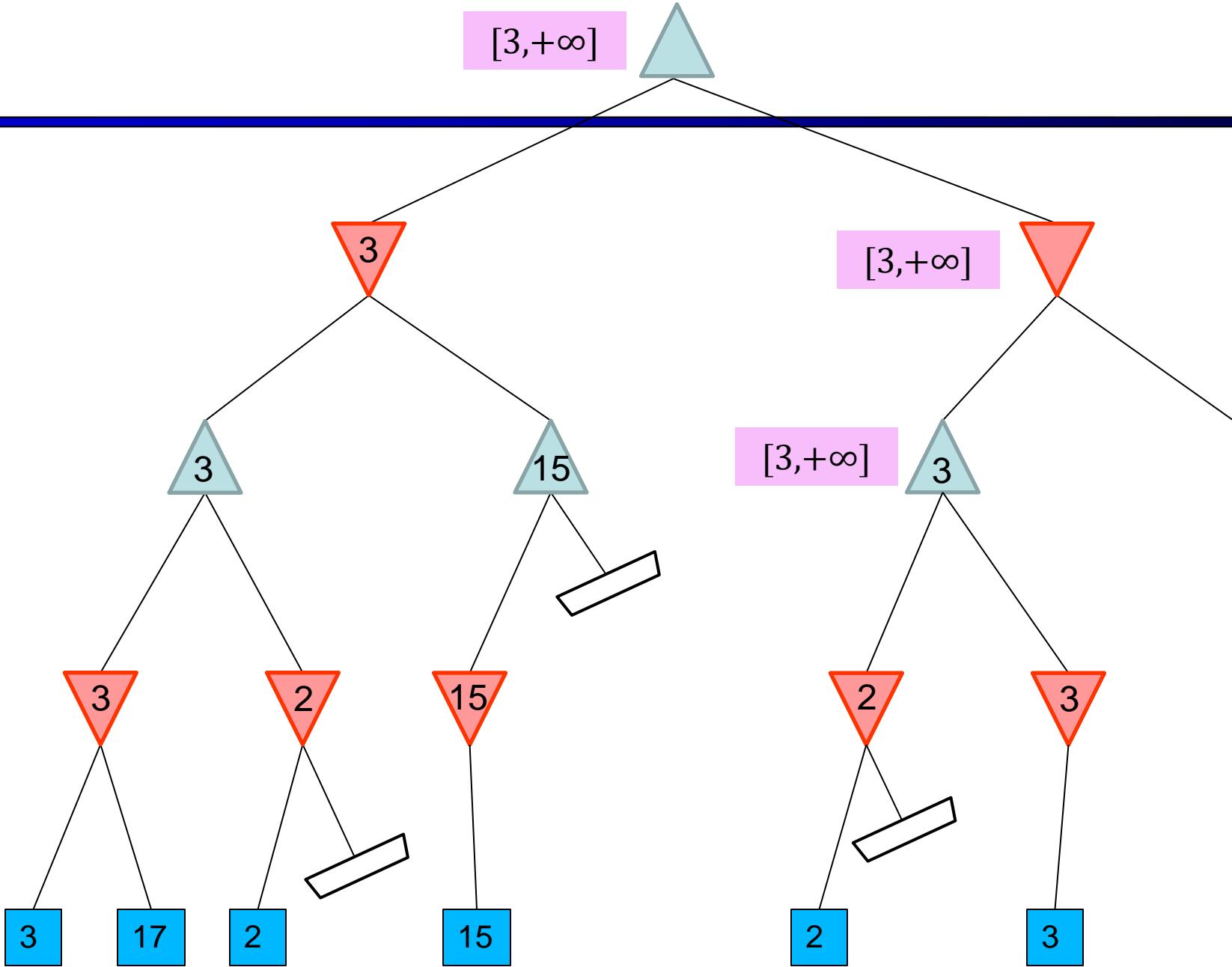


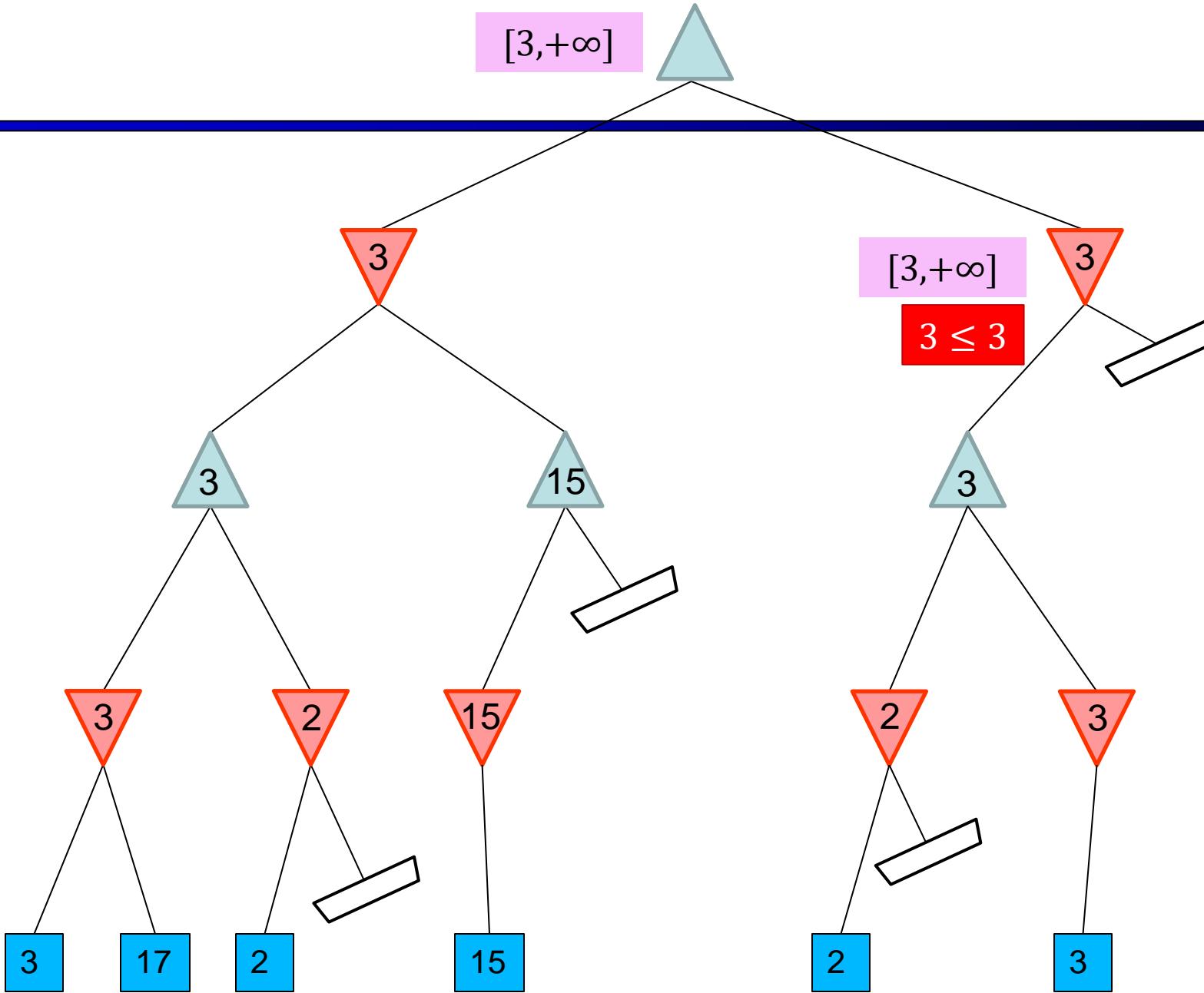


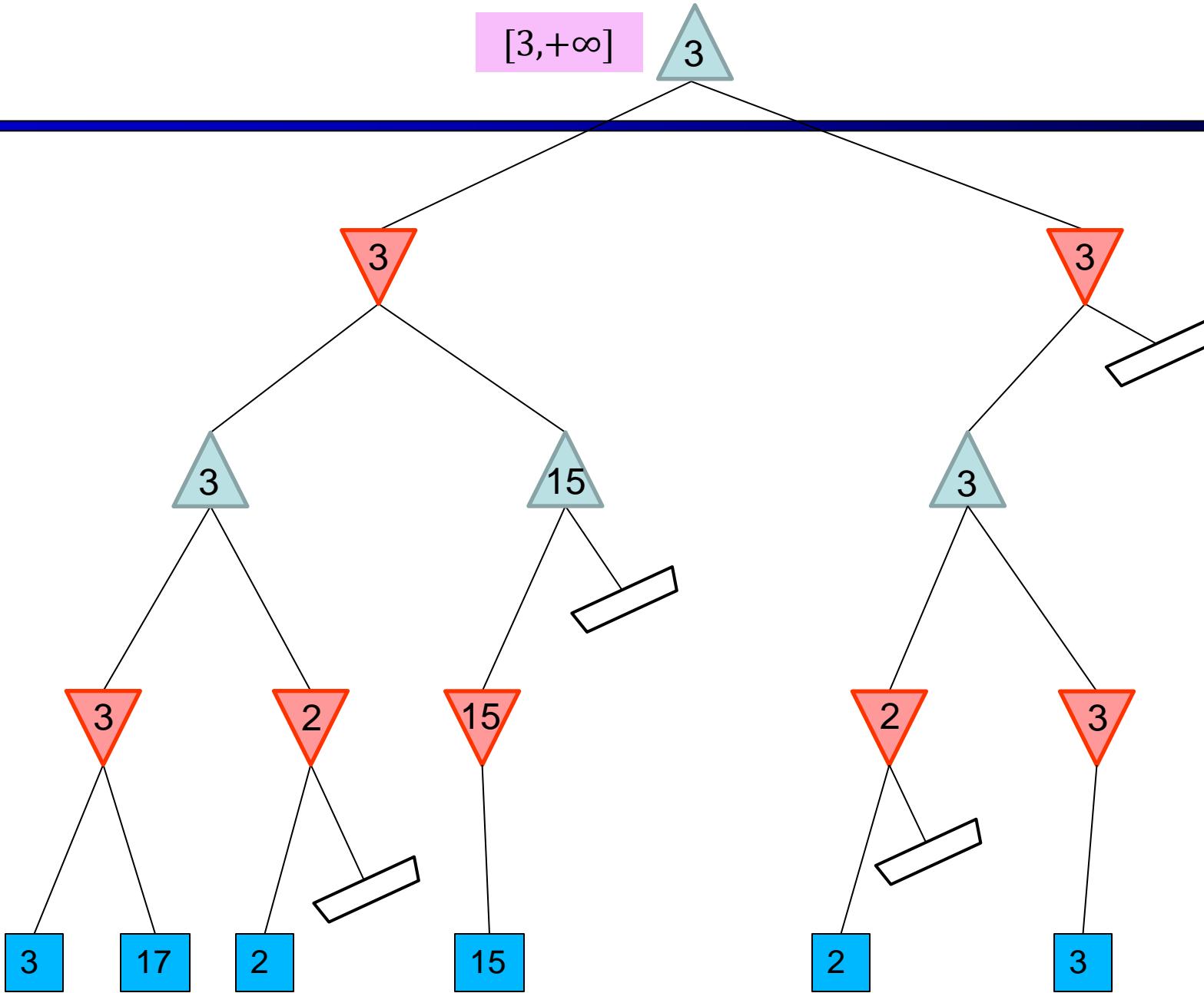










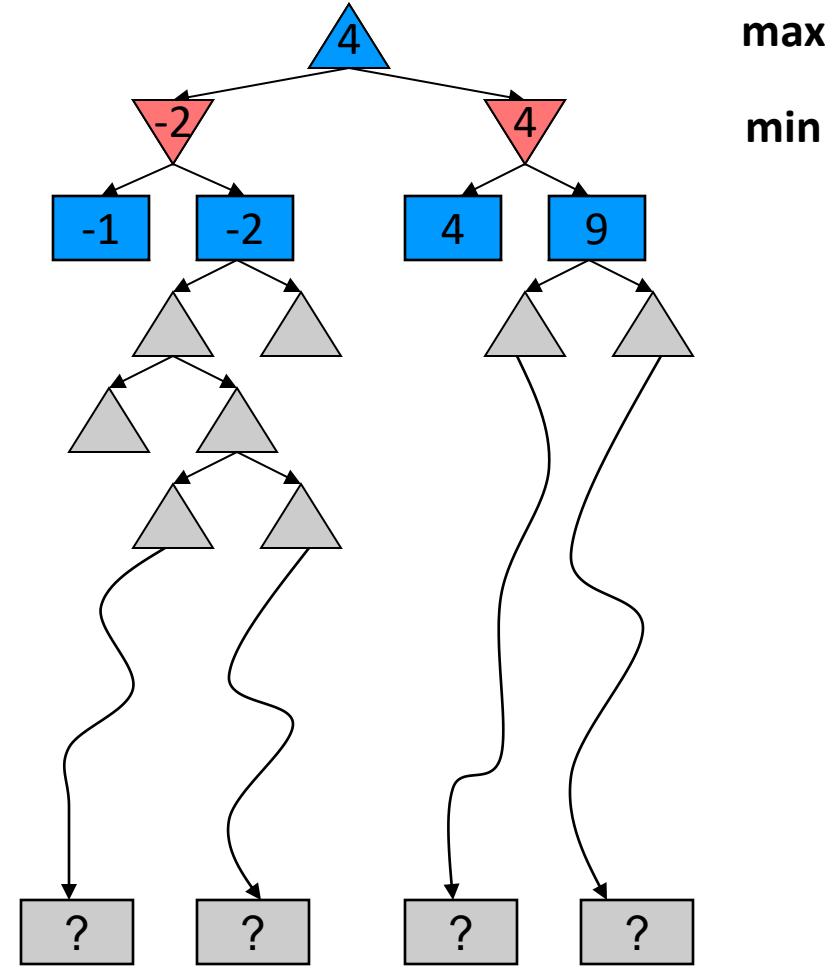


Erőforrások limitációja



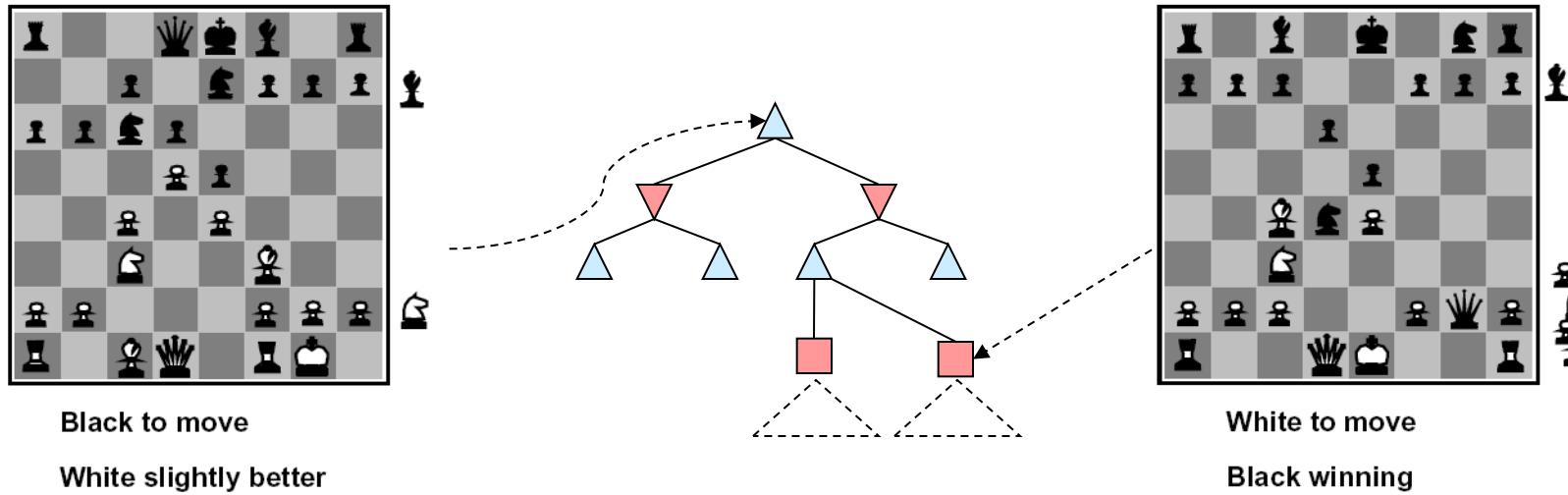
Erőforrások limitációja

- Probléma: a realisztikus játékokban a keresés nem tud eljutni a levélcsomópontokig!
- Megoldás: Mélységkorlátosított keresés
 - A teljes mélység helyett csak egy korlátos mélységig menjünk le a fában
 - A végállapotok hasznosságát cseréljük le egy kiértékelő függvénytel (állapot -> szám érték)
- Példa (sakk):
 - Tegyük fel, hogy 100 másodpercünk van minden lépésnél, és 10 ezer csomópontot tudunk megvizsgálni másodpercenként
 - Tehát lépésenként 1 millió állapotot tudunk megvizsgálni
 - α - β nyesés esetén az elérhető mélység: 8 – elég jó sakkprogram
- Az optimális játék már nem garantált
- Ha bármikor le kell tudni állítani: iteratívan mélyülő keresés



Kiértékelő függvények

- A kiértékelő függvények nem végállapot csomópontokat értékelnek ki



- Ideális függvény: az adott állapot aktuális minimax értékét adja vissza
- A gyakorlatban: tipikusan jegyek súlyozott lineáris kombinációja

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

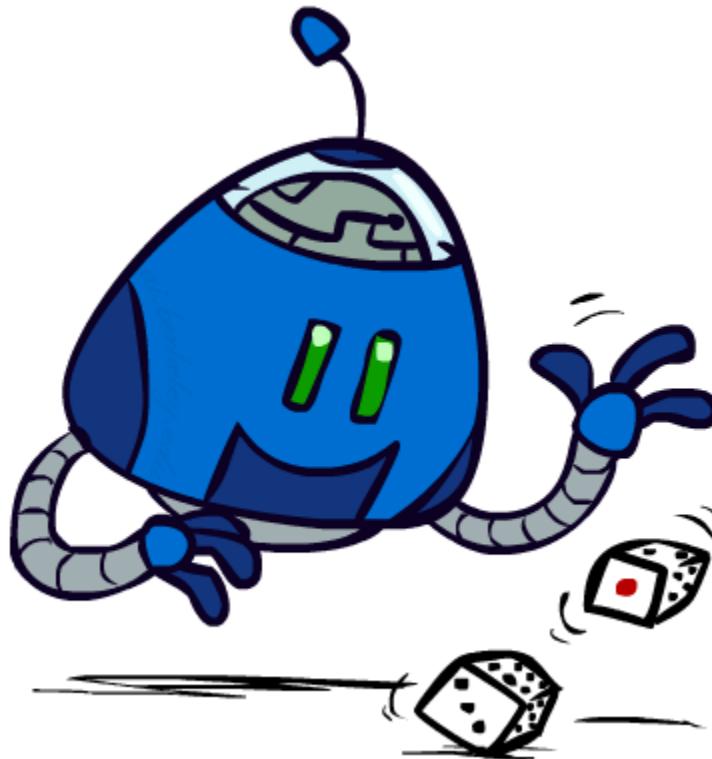
- pl.: $f_1(s) = (\text{világos vezérek száma} - \text{sötét vezérek száma})$, stb.

A mélység számít!

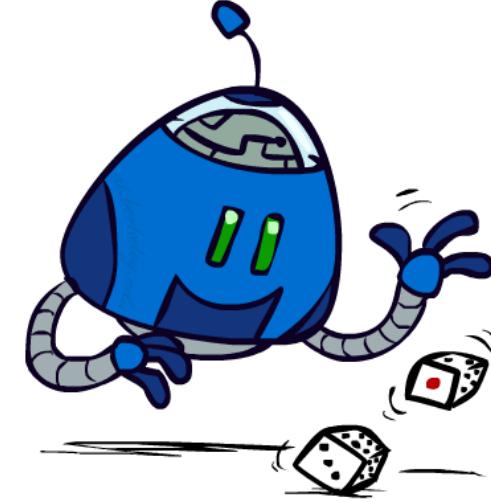
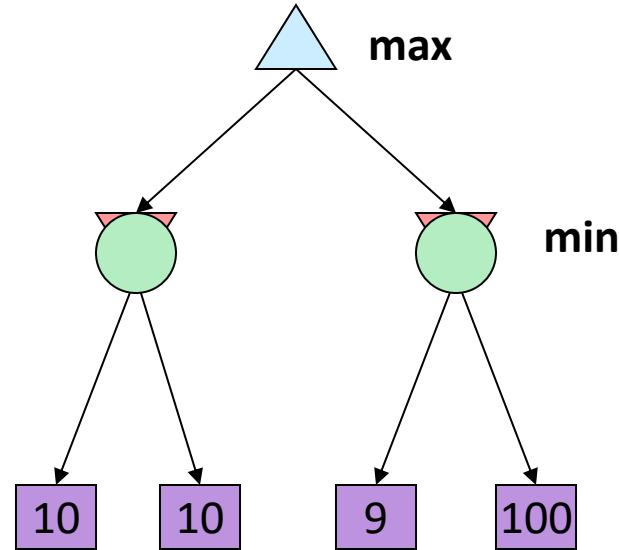
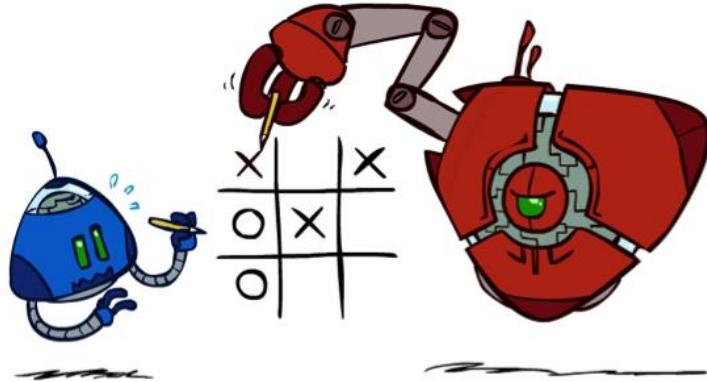
- A kiértékelő függvények mindenkor pontatlanok
- Minél mélyebben van a kiértékelő függvény, annál kevésbé számít a pontossága
- A kiértékelő függvény komplexitása vs. a keresés (számítás) komplexitása



Bizonytalan kimenetelek



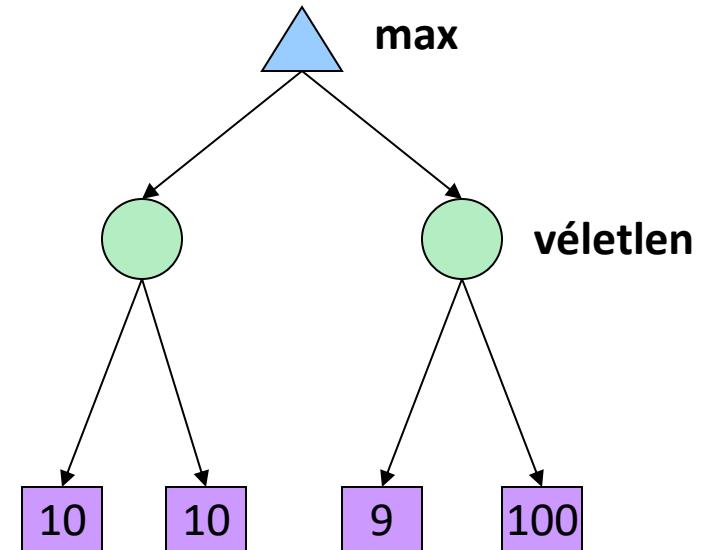
Legrosszabb eset vs. átlagos eset



Ötlet: Bizonytalan kimeneteleket a véletlen irányítja, nem egy ellenfél!

Expectimax Keresés

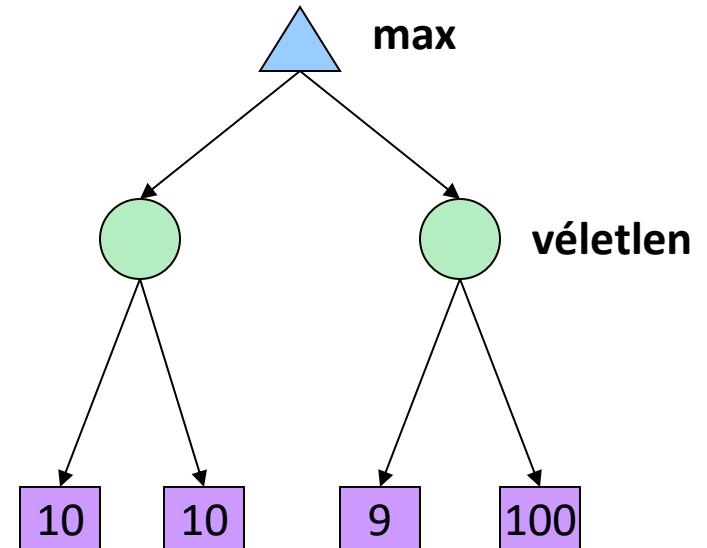
- Miért nem tudjuk (mindig), hogy egy cselekvésnek mi a kimenetele?
 - Explicit véletlen: kockadobás
 - Véletlenszerűen viselkedő ellenfelek: PacMan szellemek véletlen mozgása
 - Cselekvés nem sikerül: robot mozgásánál megcsúszik a kerék
- Az értékeknek az átlagos kimenetelt kellene tükrözniük (expectimax), nem pedig a lehető legrosszabb (minimax) kimenetelt



Expectimax Keresés

- **Expectimax keresés:** számítsuk ki az átlagos pontszámot (hasznos) optimális játékot feltételezve

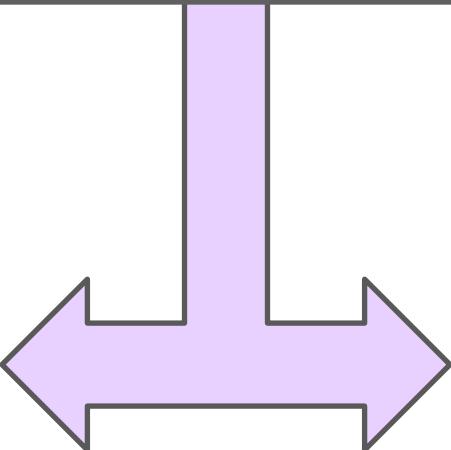
- **Max** csomópontok, mint minimax keresés esetében
- Véletlenszerű csomópontok, mint **Min** csomópontok, de a kimenetel bizonytalan
- Számítsuk ki a **várható hasznosságot**
- Azaz a gyermekcsomópontok súlyozott átlagát (várhatóértékét)



Expectimax Pszeudokód

```
def value(state):
    if the state is a terminal state: return the state's utility
    if the next agent is MAX: return max-value(state)
    if the next agent is EXP: return exp-value(state)
```

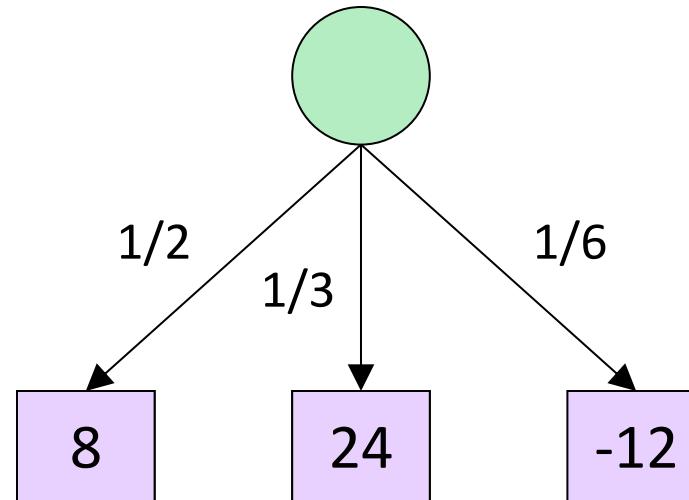
```
def max-value(state):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor))
    return v
```



```
def exp-value(state):
    initialize v = 0
    for each successor of state:
        p = probability(successor)
        v += p * value(successor)
    return v
```

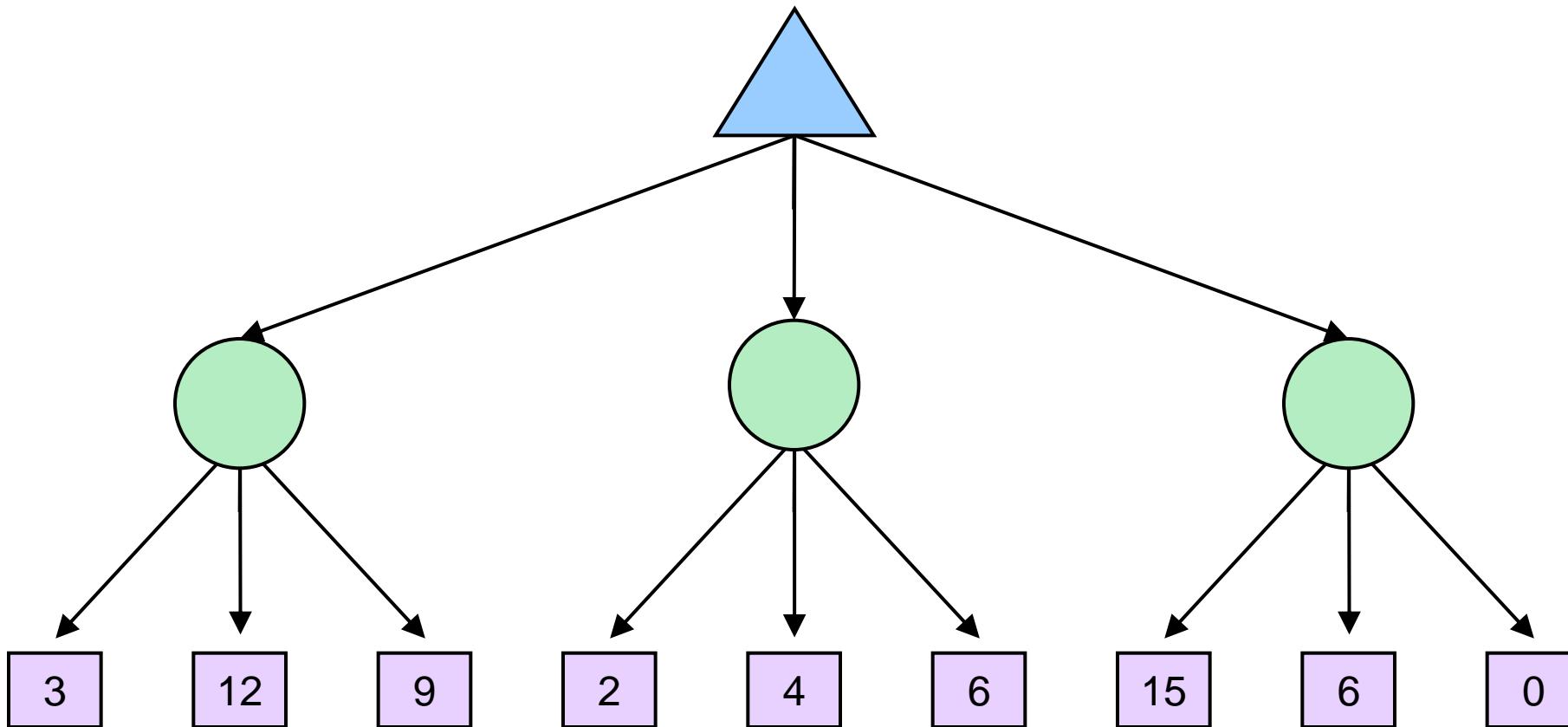
Expectimax Pszeudokód

```
def exp-value(state):  
    initialize v = 0  
    for each successor of state:  
        p = probability(successor)  
        v += p * value(successor)  
    return v
```

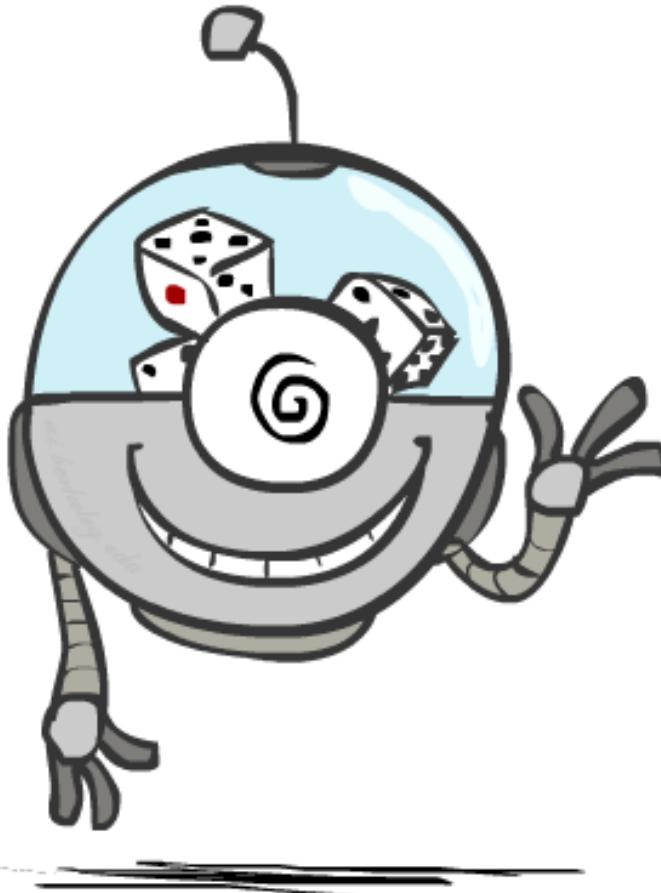


$$v = (1/2) (8) + (1/3) (24) + (1/6) (-12) = 10$$

Expectimax Példa

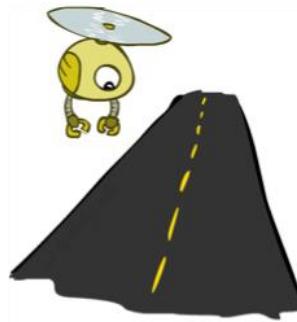


Valószínűségek



Emlékeztető: valószínűségek, várható érték

- A véletlen változó regy eseményt reprezentál, melynek kimenetele nem ismert
- A valószínűségi eloszlás voltaképpen súlyok hozzárendelése kimenetelekhez
- Példa: Forgalom az autópályán
 - Véletlen változó: T = Forgalom nagysága
 - Kimenetelek (értékek): $T \in \{\text{nincs}, \text{kicsi}, \text{nagy}\}$
 - Eloszlás: $P(T=\text{nincs}) = 0.25$, $P(T=\text{kicsi}) = 0.50$, $P(T=\text{nagy}) = 0.25$
- Valószínűségi axiómák:
 - A valószínűségek nem negatívak
 - minden lehetséges kimenetel összesített valószínűsége 1-et ad



0.25



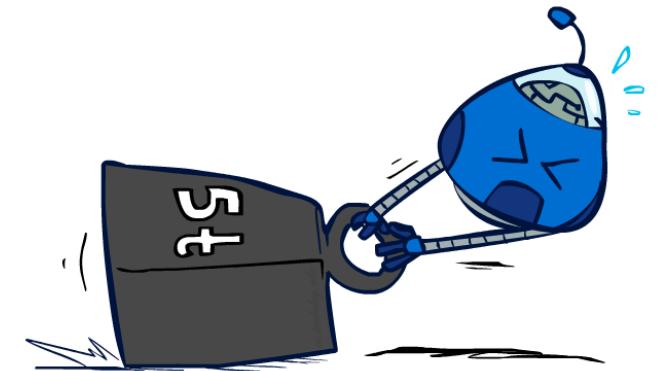
0.50



0.25

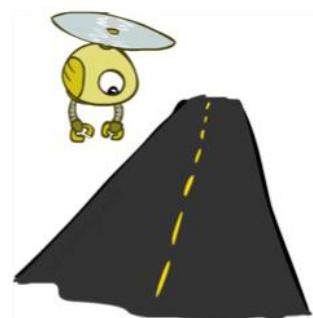
Emlékeztető: valószínűségek, várható érték

- Egy véletlen valószínűségi változó várható értéke a kimenetelek valószínűségi eloszlás szerint súlyozott átlaga
- Példa: Átlagosan mennyi idő kijutni a reptérre?



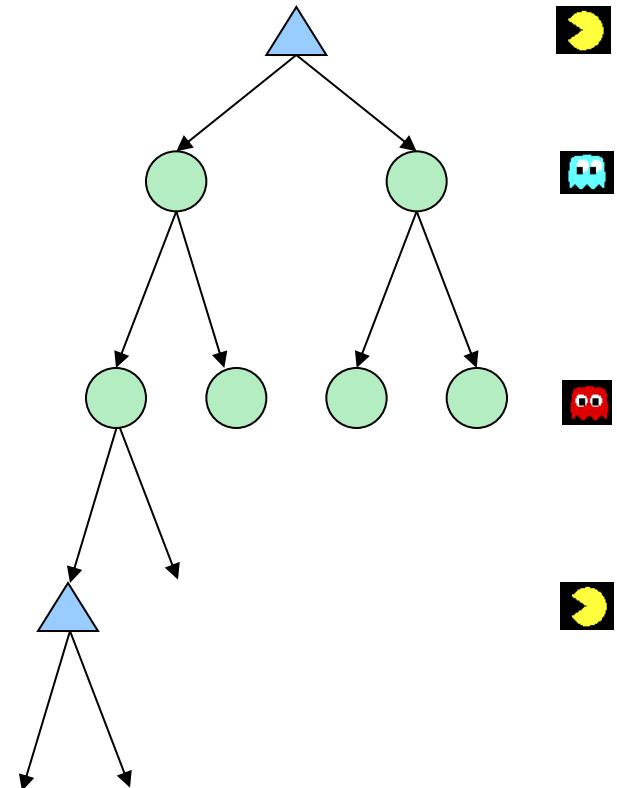
Idő:	20 min	x	+	30 min	x	+	60 min	x
Valószínűség:	0.25			0.50			0.25	

35 min



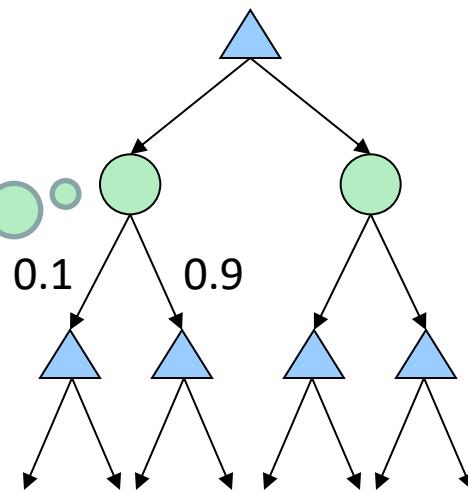
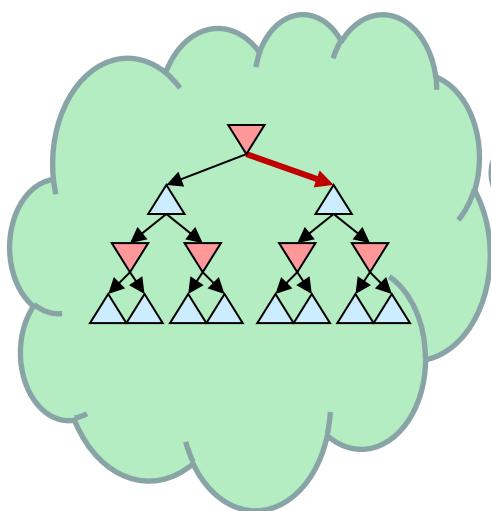
Milyen valószínűségeket használunk?

- Az expectimax keresésnél van egy valószínűségi modellünk arról, hogy az ellenfél (vagy a környezet) hogyan fog viselkedni tetszőleges állapotban
 - A modell lehet egy egyszerű egyenletes eloszlás (kockadobás)
 - A modell lehet kifinomult és nagy számításigényű
 - A modellben vannak véletlen csomópontok minden olyan kimenetelre, amelyet nem tudunk kézben tartani (ellenfél vagy környezet)
 - A modell lehetővé teheti az ellenfelek ellenséges akciójait
- Egyelőre tegyük fel, hogy a véletlen csomópont együtt jön létre az kimenetelek eloszlását leíró valószínűségekkel



Informált valószínűségek

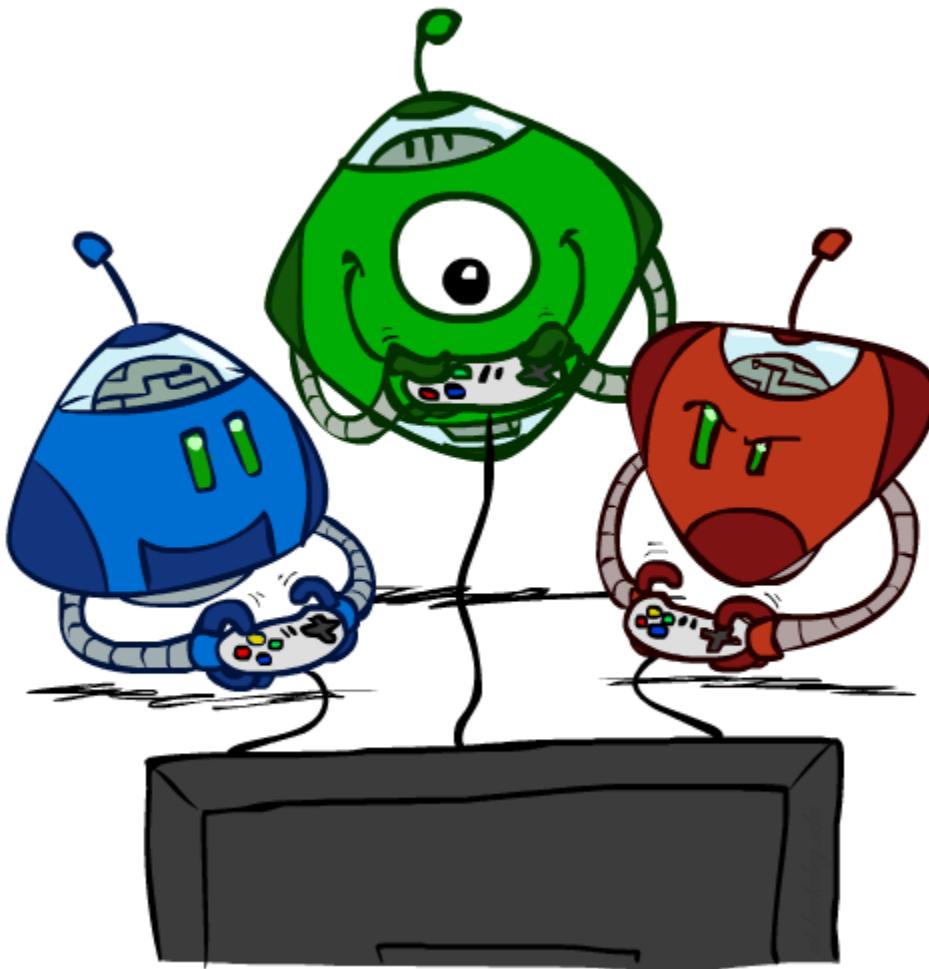
- Tegyük fel, hogy az ellenfél ténylegesen egy 2 mélységű minimaxot futtat, amelynek eredményét az esetek 80%-ban használja fel, a többiben véletlenszerűen lép
- Kérdés: Milyen fakeresést használjunk ehhez?



- **Válasz: Expectimax!**

- Hogy minden egyes véletlen csomópont valószínűségét ki tudjuk számítani, le kell futtatni az ellenfél szimulációját
- Ez viszonylag hamar válik nagyon lassúvá
- Még rosszabb, ha azt is le kell szimulálni, hogy az ellenfél szimulál minket...
- Kivéve a minimaxot, ami összesugorodik egyetlen keresési fába

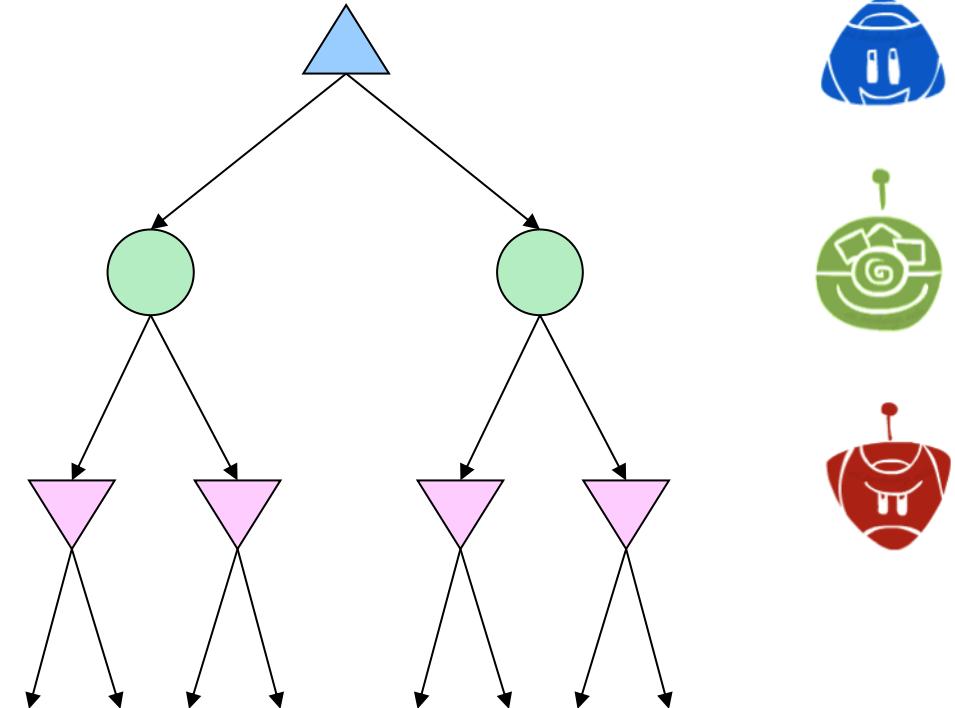
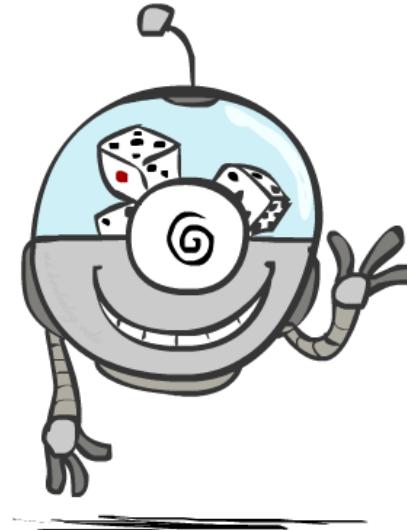
További játéktípusok



Kevert rétegű játék

■ Expectiminimax

- Környezet plusz egy véletlenszerű ágens, aki minden min/max játékos után lép
- minden csomópont kiszámítja a „megfelelő kombinációját” a gyermekéinek



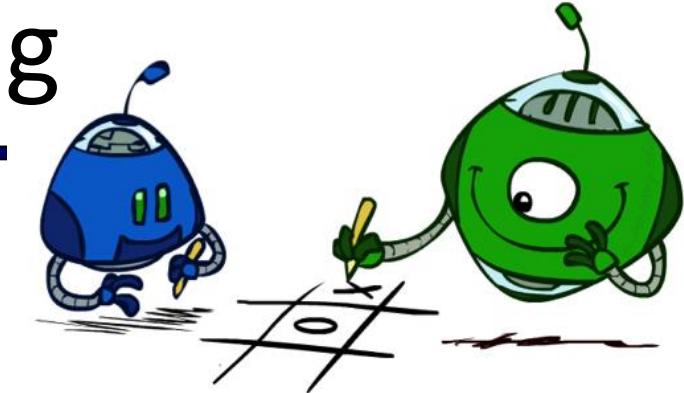
Példa: Ostábla(backgammon)

- Kockadobások növelik a b-t : 21 lehetséges dobás 2 d6 kockával
 - Ostábla(Backgammon) \approx 20 legális lépés
 - Mélység 2 = $20 \times (21 \times 20)^3 = 1.2 \times 10^9$
- Ahogy a mélység növekszik annak a valószínűsége, hogy érintünk egy adott csomópontot csökken
 - A keresés hasznossága csökken
 - Ezért limitálni a mélységet kevésbé problémás
 - A vágás viszont nehéz...
- TDGammon: 2 mélységű keresés + jó kiértékelő függvény + megerősítéses tanulás
=> világbajnoki szintű játékos



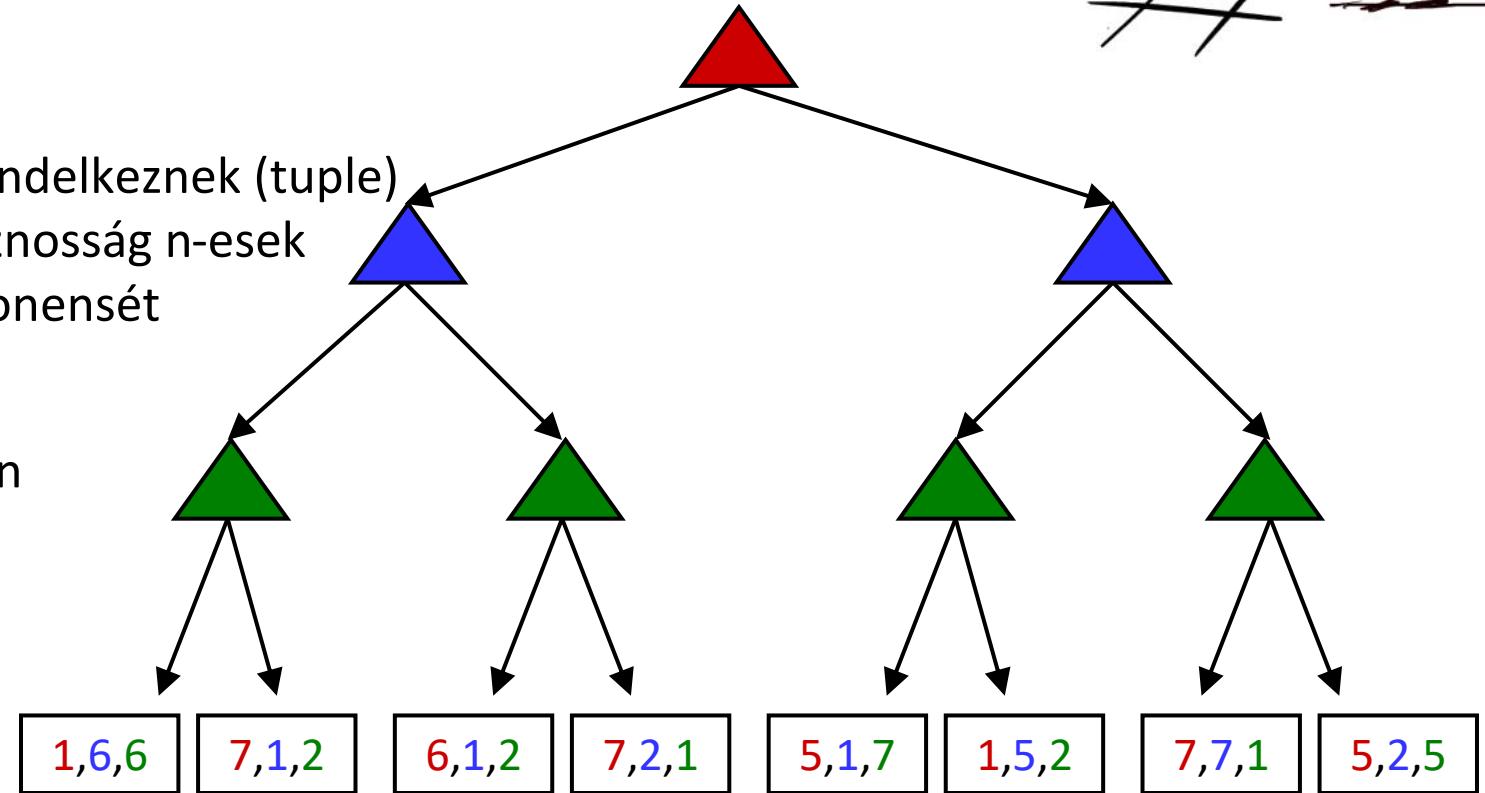
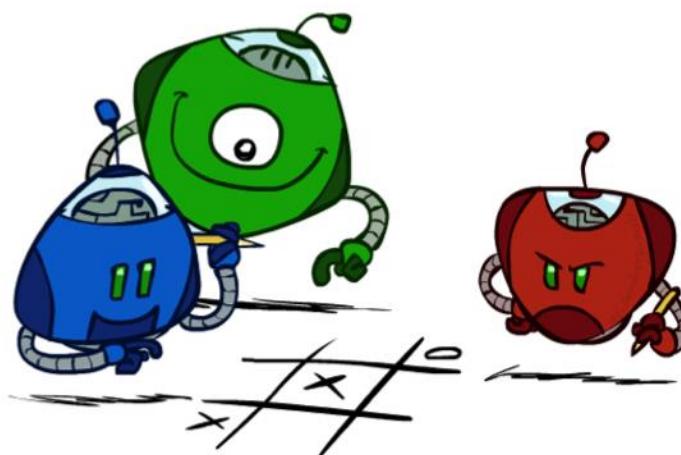
Több ágenses hasznosság

- Mi van abban az esetben, ha nem zéró összegű a játék, hanem több játékos van?



- Minimax általánosítása:

- Levelek hasznosság n-esekkel rendelkeznek (tuple)
- Csomópont értékek szintén hasznosság n-esek
- Mindegyik játékos a saját komponensét maximalizálja
- Ebből adódhat kooperatív és versengő viselkedés dinamikusan



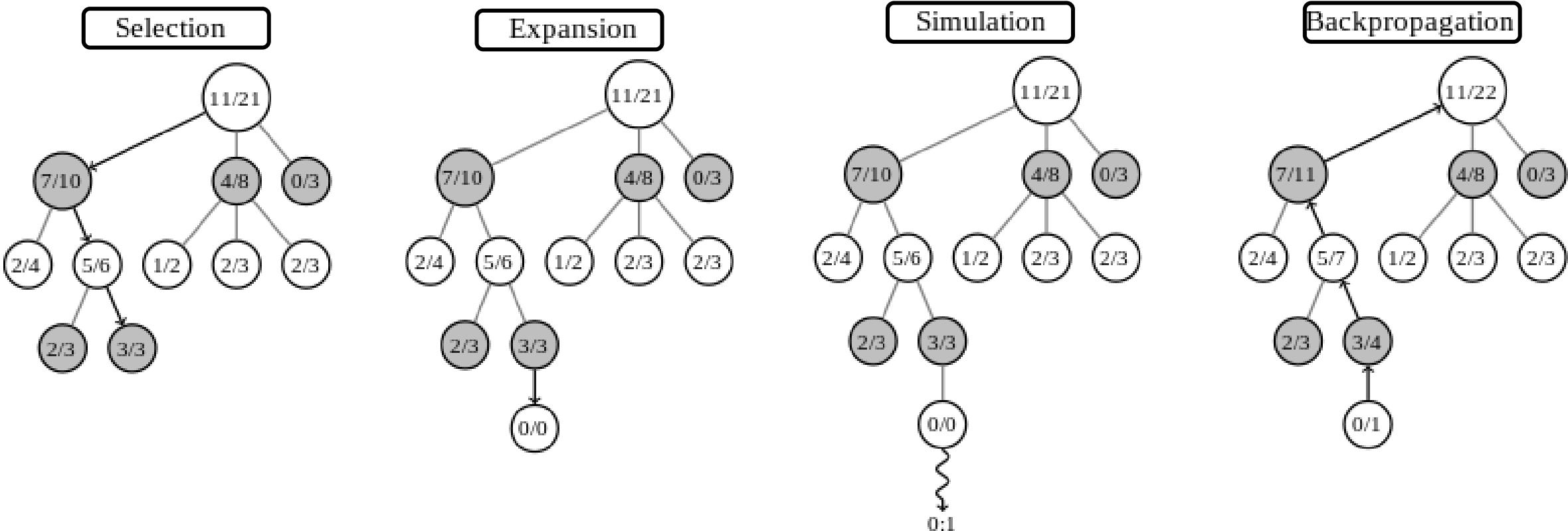
Monte Carlo Tree Search (MCTS)

- Monte Carlo kísérletek :
 - ismételt véletlen mintavételel (egy adott eloszlásból/mintahalmazból)
 - összegyűlt minták gyakorisága (aránya) alapján közelíthetők numerikusan nehezen számítható értékek
- MCTS: Monte Carlo mintavételezést használó fa keresés
- Jól használható komplex döntések meghozatalához
- A legerősebb GO MI-k (Fuego, Pachi, Zen, and Crazy Stone) minden MCTS-en alapulnak

Monte Carlo Tree Search lépései

MCTS minden ciklusa az alábbi négy lépésből áll

1. Kiválasztás (selection)
2. Terjeszkedés (expansion)
3. Szimuláció (simulation)
4. Visszaterjesztés (backpropagation)

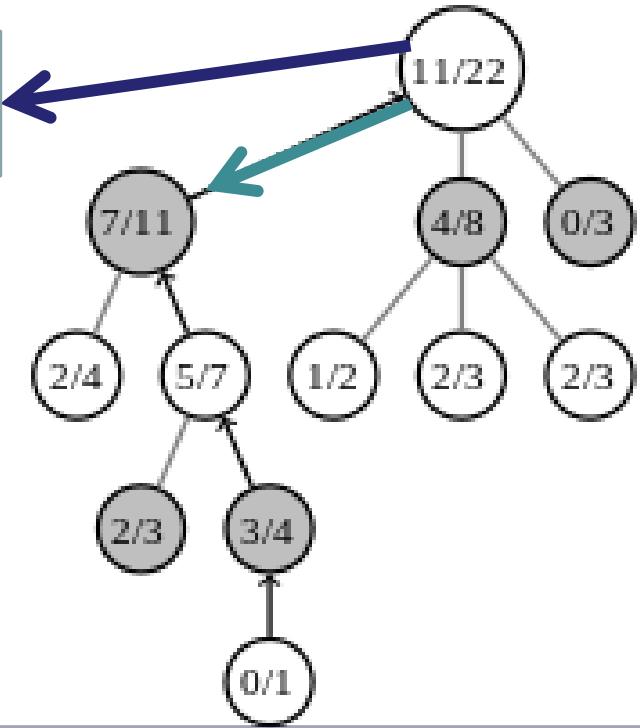


MCTS – Tic-Tac-Toe

- <https://www.youtube.com/watch?v=ghhznqBoESY>
- <https://vgarciasc.github.io/mcts-viz/>

MCTS – Upper Confidence Bounds (UCB)

- Felfedezés (exploration): új még meg nem látogatott csomópont kiválasztása
- Kiszákmányolás(exploitation): már meglátogatott csomópont kiválasztása (pontosság növelése)
- Felfedezés és kiszákmányolás egyensúlya a cél
 - Kocsis, L. & Szepesvári, C.: Bandit based Monte-Carlo planning (2006) -> UCB
 - Optimális megoldáshoz konvergál



$$\frac{w_i}{n_i} + C \cdot \sqrt{\frac{\ln(t)}{n_i}}$$

Exploitation

Exploration

w_i #győzelmek száma i csomópont meglátogatása után
 n_i #i csomópont meglátogatásainak száma
C felfedezési paraméter
 t #i csomópont szülei meglátogatásainak száma