Kétszemélyes játékok

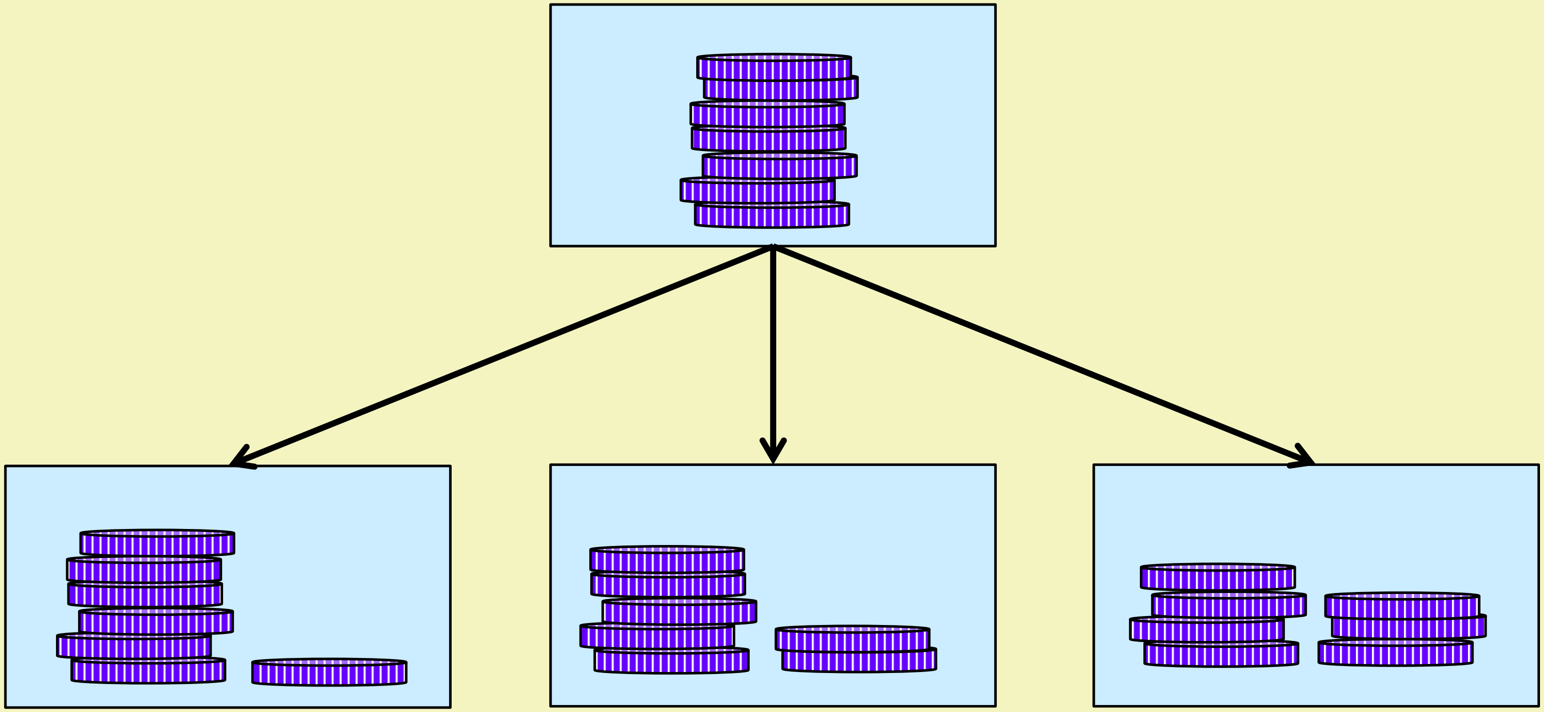
Kétszemélyes, teljes információjú, véges, determinisztikus,zéró összegű játékok

- Két játékos lép felváltva adott szabályok szerint, amíg a játszma véget nem ér.
- Mindkét játékos ismeri a maga és az ellenfele összes múltbeli és jövőbeli lépéseit és lépési lehetőségeit, és azok következményeit.
- Minden lépés véges számú lehetőség közül választható, és minden játszma véges lépésben véget ér. Egy lépés determinisztikus, a véletlennek nincs szerepe.
- Amennyit a játszma végén az egyik játékos nyer, annyit veszít a másik. (Legegyszerűbb változatban két esélyes: egyik nyer, másik veszít; vagy három esélyes: döntetlen is megengedett)

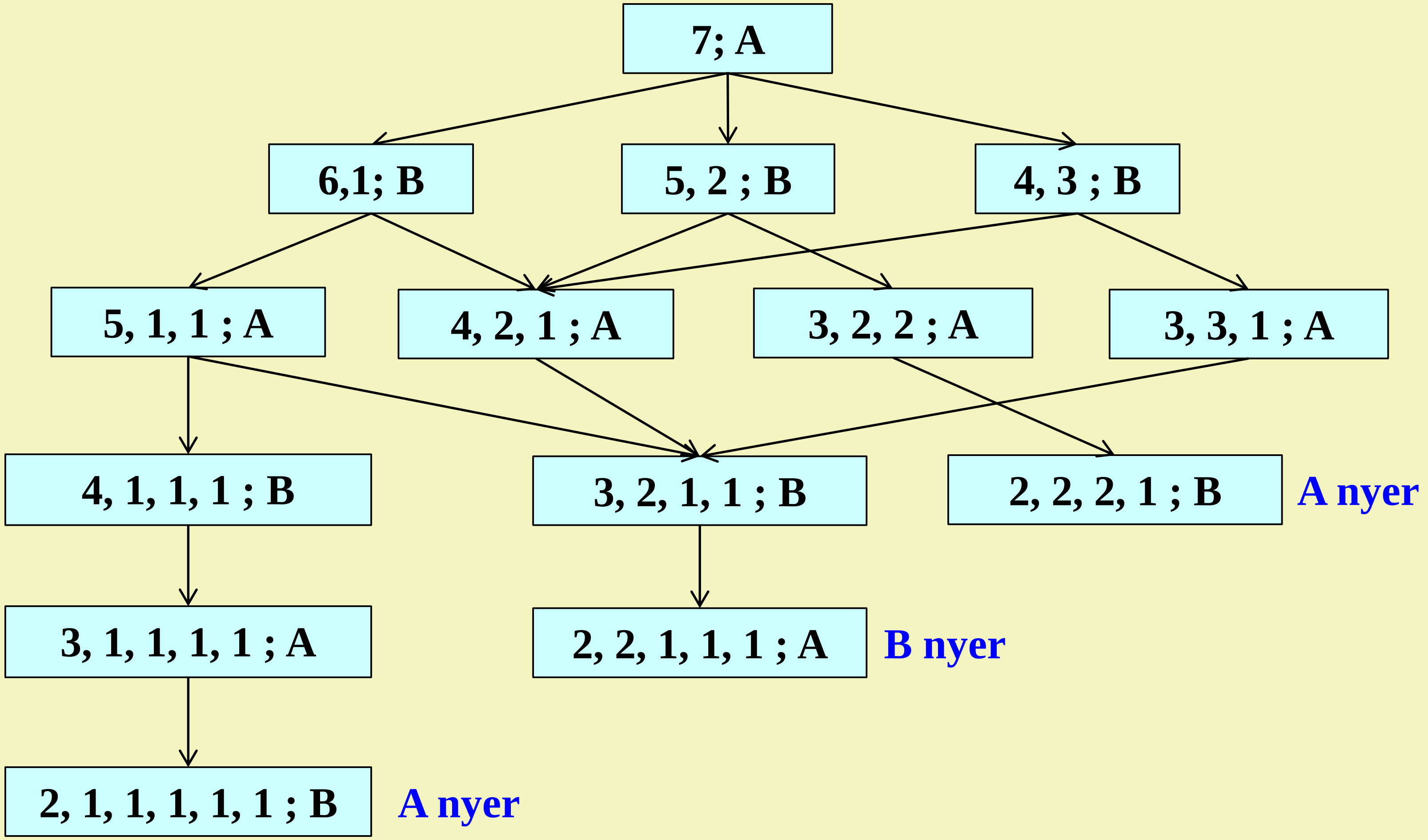
Állapottér modell

- állapot – állás + soron következő játékos
- művelet – lépés
- kezdő állapot – kezdőállás + kezdő játékos
- végállapot – végállás + játékos
- payoff függvény: p_A, p_B : végállapot$\longrightarrow \mathbb{R}$ (játékosok: A, B)
  - Zéró összegű kétszemélyes játékban:
    $p_A(t) + p_B(t) = 0$ minden t végállapotra
  - Speciális esetben (a továbbiakban ezt fektételezzük):
    - $p_A(t) = +1$ ha A nyer
    - $p_A(t) = -1$ ha A veszít
    - $p_A(t) = 0$ ha döntetlen

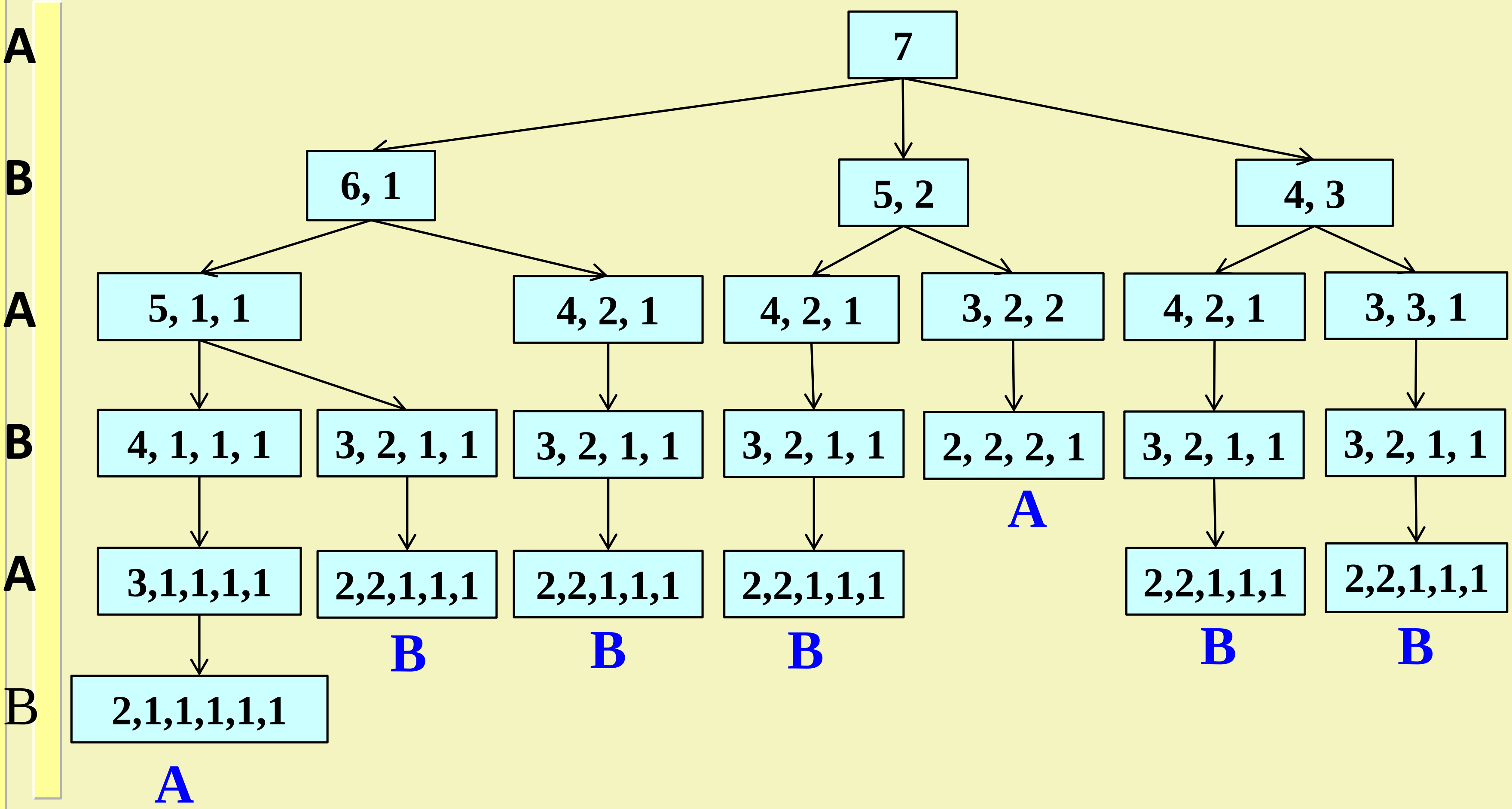
Grundy mama játéka



Grundy mama állapot-gráfja



Grundy mama játékfája

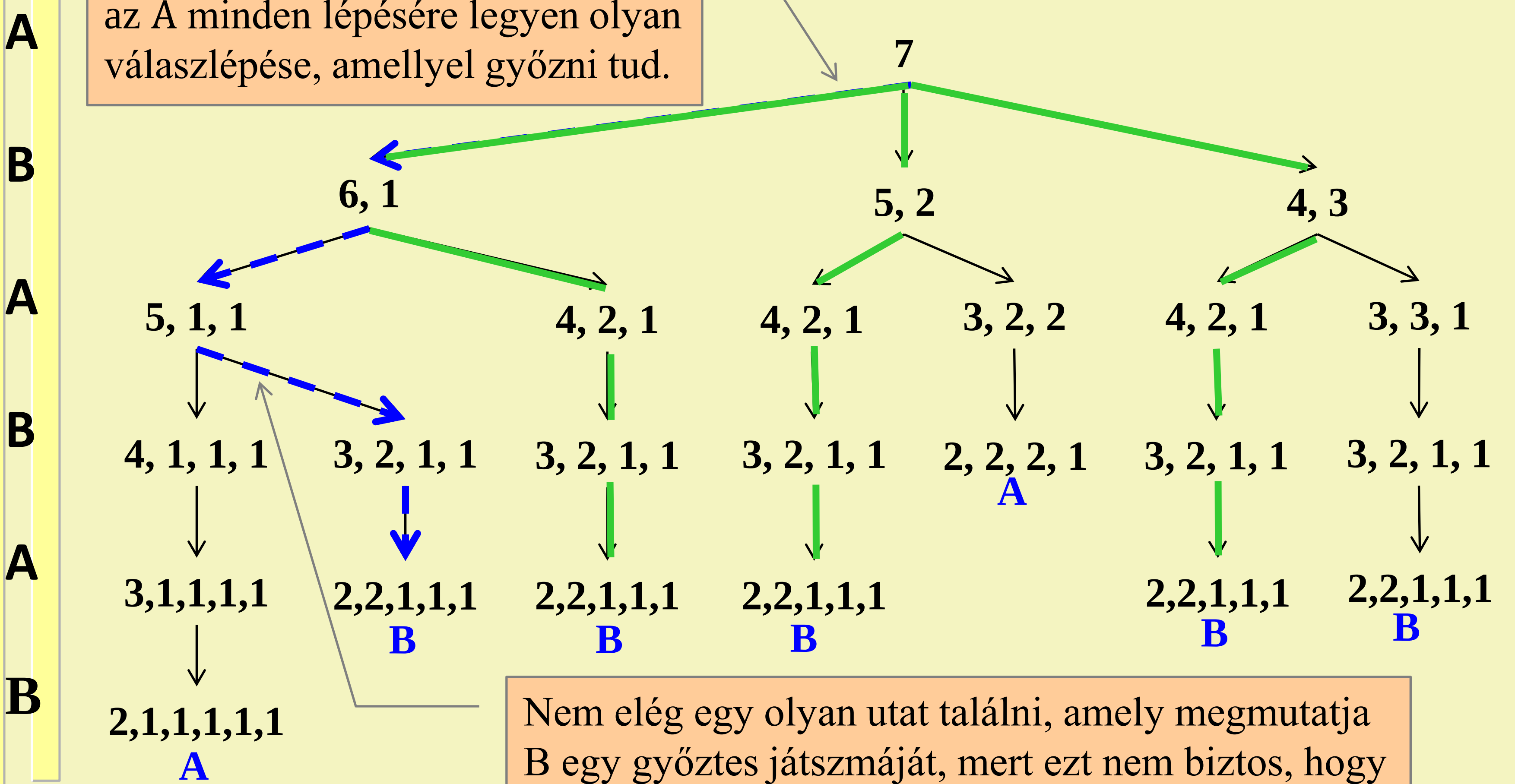


Játékfa

- csúcs – állás (egy állás több csúcs is lehet)
- szint – játékos (felváltva az A és B szintjei)
- él – lépés (szintről szintre)
- gyökér – kezdőállás (kezdő játékos)
- levél – végállások
- ág – játszma

Hogyan tud a B játékos biztosan nyerni?

A B-nek arra van szüksége, hogy az A minden lépésére legyen olyan válaszlépése, amellyel győzni tud.



Nem elég egy olyan utat találni, amely megmutatja B egy győztes játszmáját, mert ezt nem biztos, hogy B végig tudja játszani az A válaszlépései miatt.

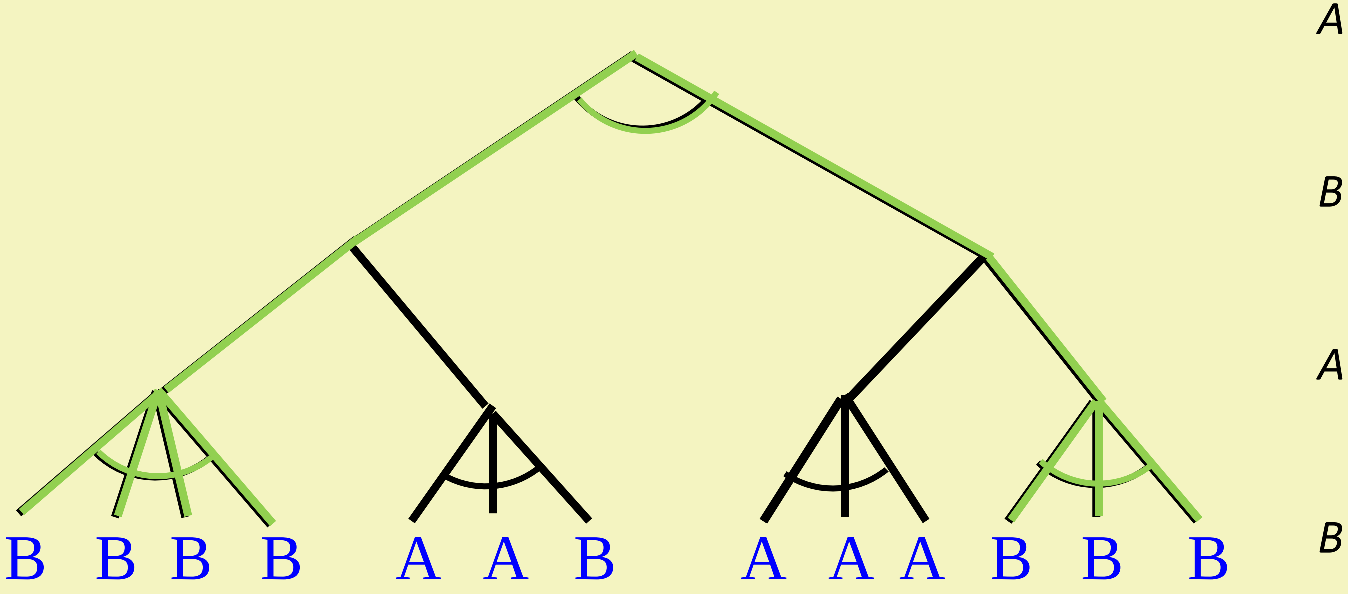
Nyerő stratégia

- Egy játékos nyerő stratégiája egy olyan elv, amelyet betartva az ellenfél minden lépésére tud olyan választ adni, hogy megnyerje a játékot.
- A nyerő stratégia NEM egyetlen győztes játszma, hanem olyan győztes játszmák összessége, amelyek közül az egyiket biztos végig tudja játszani az a játékos, aki rendelkezik a nyerő stratégiával.
- Hasznos lehet a nem-vesztő stratégia megtalálása is, ha döntetlent is megengedő játéknál nincs győztes stratégia.
- Általános zéró összegű játékoknál beszélhetünk adott hasznosságot biztosító stratégiáról.

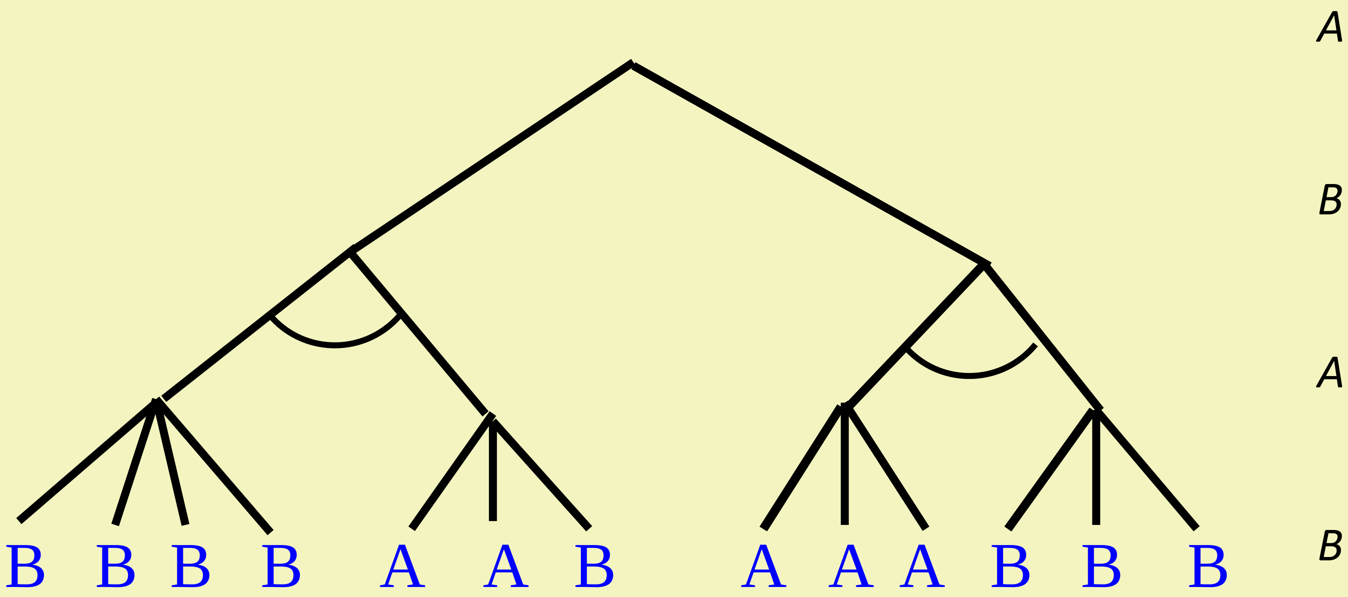
Megjegyzés

- A játék az egyik játékos szempontjából egy ÉS/VAGY fával ábrázolható.
  - saját szinten egy csúcs utódai között VAGY kapcsolat van
  - ellenfél szintjén egy csúcs utódai között ÉS kapcsolat van
- A nyerő (nem-vesztő) stratégiát az ÉS/VAGY játékfa azon hiper-útja mutatja, amely a gyökércsúcsból csupa nyerő (nem-vesztő) levélcsúcsba vezet.
- A nyerő stratégia keresése tehát egy ÉS/VAGY fabeli hiper-út keresési probléma.

*Nyerő stratégia keresése a **B** játékos ÉS/VAGY fájában*



Nyerő stratégia keresése az A játékos ÉS/VAGY fájában

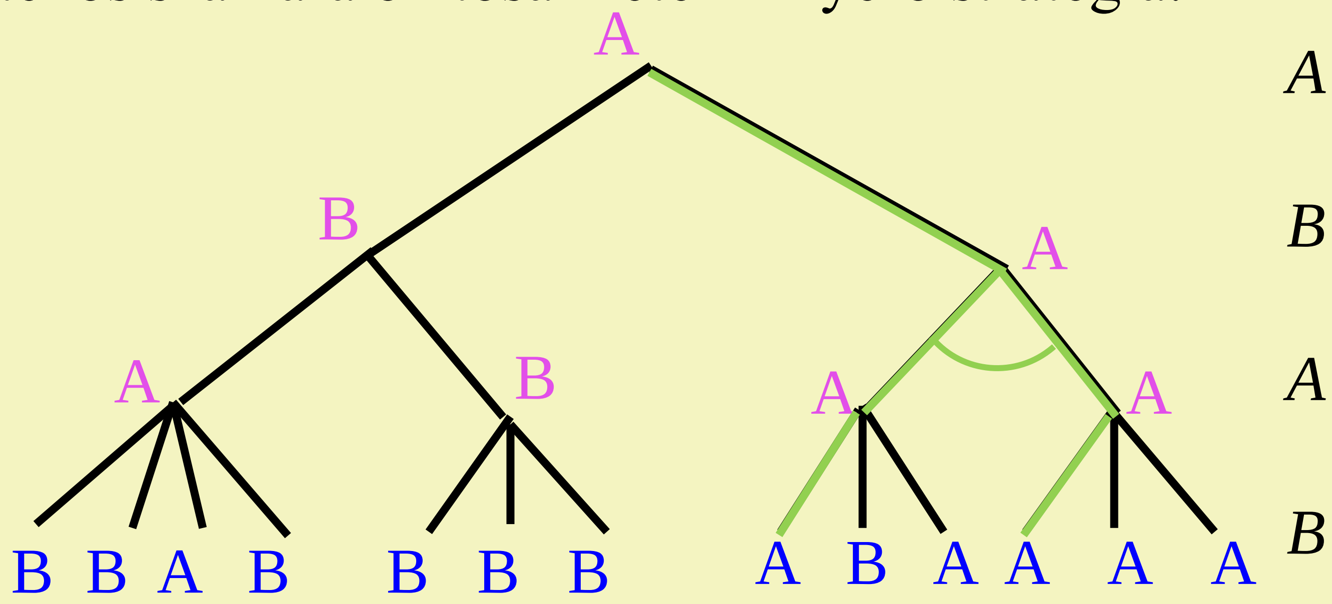


Nincs nyerő stratégia.

Csak az egyik játékosnak lehet nyerő stratégiája.

Tétel

- A két esélyes (győzelem vagy vereség) teljes információjú véges determinisztikus kétszemélyes játékokban az egyik játékos számára biztosan létezik nyerő stratégia.



- A három esélyes játékokban (van döntetlen is) a nem vesztő stratégiát lehet biztosan garantálni.

Részleges játékfa-kiértékelés

- A nyerő vagy nem-vesztő stratégia megkeresése egy nagyobb játékfa esetében reménytelen.
- Az optimális lépés helyett a soron következő jó lépést keressük.
  - Legyen a bennünket képviselő játékos neve mostantól MAX, az ellenfélé pedig MIN.
- Ehhez az aktuális állapotból indulva kell a játékfa
  1. néhány szintjét felépíteni,
  2. ezen a részfa leveleinek a hasznosságát megbecsülni,
  3. majd a soron következő lépést meghatározni.

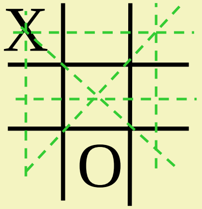
Kiértékelő függvény

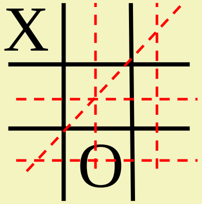
- Minden esetben szükségünk van egy olyan heurisztikára, amely a mi szempontunkból becsüli meg egy állás hasznosságát: f : Állások → [-1000 , 1000] függvény.
- Példák:
  - Sakk: (kiértékelő függvény a fehérnek)

    $f(s)$ = (fehér királynő száma) – (fekete királynő száma)
  - Tic-tac-toe: $f(s) = M(s) - O(s)$

    $M(s)$ = a saját lehetséges győztes vonalaink száma

    $O(s)$= az ellenfél lehetséges győztes vonalaink száma



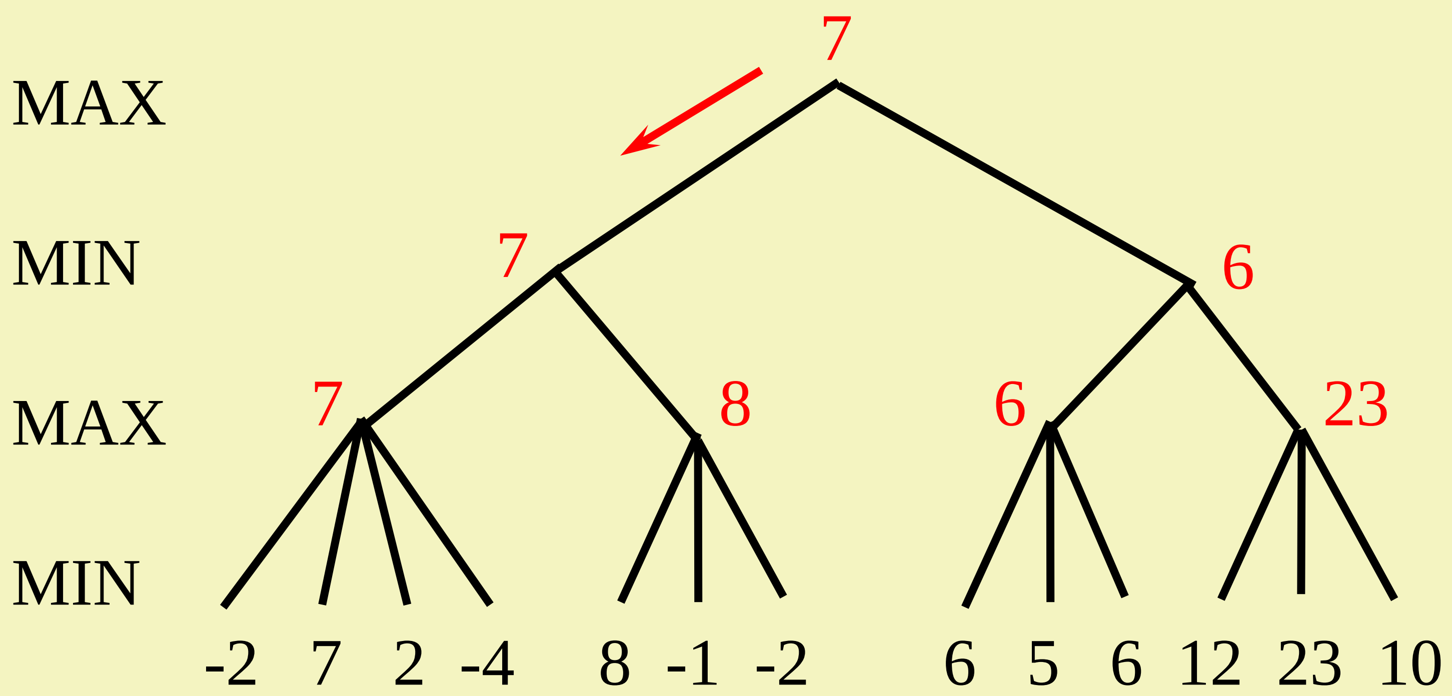


Minimax algoritmus

1. A játékfának az adott állás csúcsából leágazó részfáját felépítjük néhány szintig.
2. A részfa leveleit kiértékeljük a kiértékelő függvény segítségével.
3. Az értékeket felfuttatjuk a fában:
   - A saját (MAX) szintek csúcsaihoz azok gyermekeinek maximumát: $szülő := max\ (gyerek_1,\dots,\ gyerek_k)$
   - Az ellenfél (MIN) csúcsaihoz azok gyermekeinek minimumát: $szülő := min\ (gyerek_1,\dots,\ gyerek_k)$
4. Soron következő lépésünk ahhoz az álláshoz vezet, ahonnan a gyökérhez felkerült a legnagyobb érték.

Példa

Legyen a mi nevünk MAX, az ellenfélé MIN.



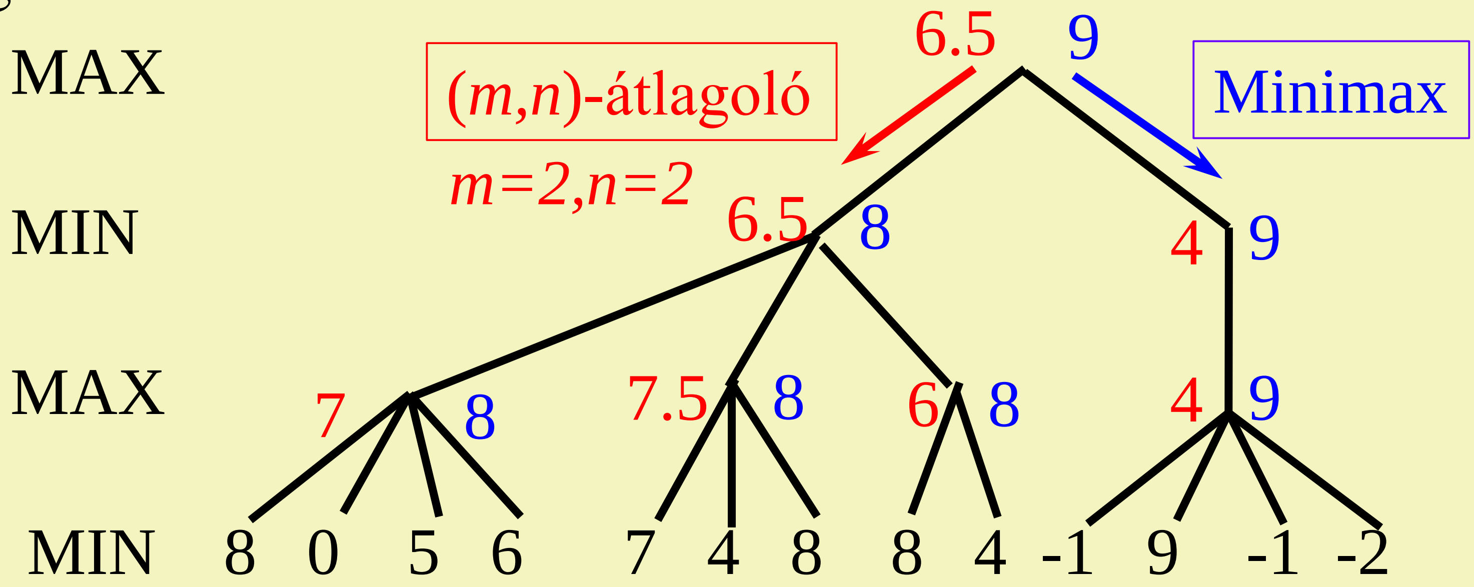
Megjegyzés

- Az algoritmust minden alkalommal, valahányszor mi következünk, megismételjük, hiszen lehet, hogy az ellenfél nem az általunk várt legerősebb lépésekkel válaszol, mert:

  - eltérő mélységű részfával dolgozik,
  - más kiértékelő függvényt használ,
  - nem minimax eljárást alkalmaz,
  - hibázik.

Átlagoló kiértékelés

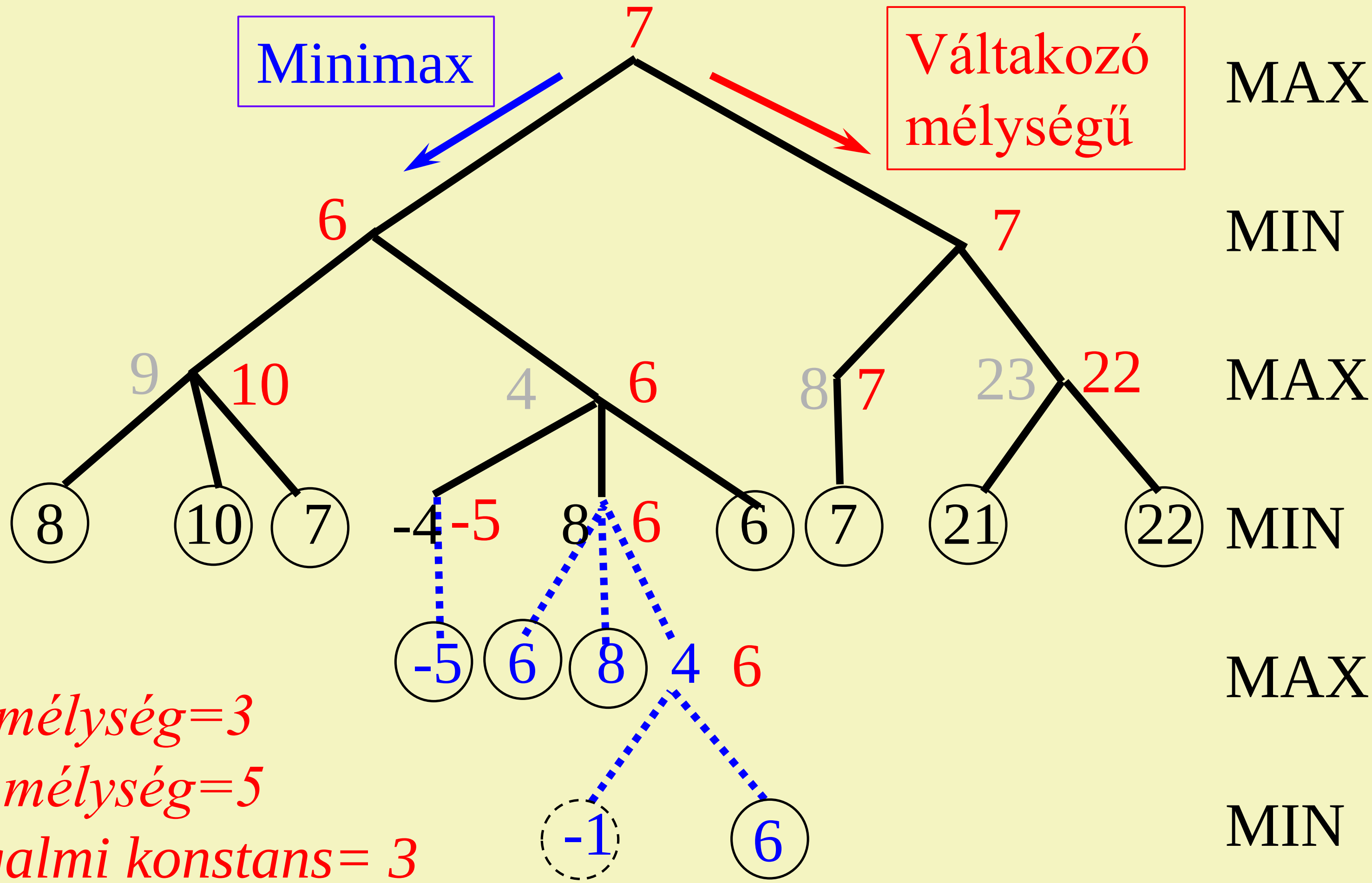
- Célja a kiértékelő függvény esetleges tévedéseinek simítása.
- MAX szintjeire az *m* darab legnagyobb értékű gyerek (max_m) átlaga, a MIN-re az *n* darab legkisebb értékű gyerek (min_n) átlaga kerül.



Váltakozó mélységű kiértékelés

- Célja, hogy a kiértékelő függvény minden ágon reális értéket mutasson. Megtévesztő lehet egy csúcsnál ez az érték ha annak szülőjénél a kiértékelő függvény lényegesen eltérő értéket mutat: a játék ezen szakasza nincs nyugalomban.
- Egy adott szintig (minimális mélység) mindenképpen felépítjük a részfát,
- majd ettől a szinttől kezdve egy másik adott szintig (maximális mélység) csak azon csúcsok gyerekeit állítjuk elő, amelyek még nincsenek nyugalomban, amelyre <u>nem</u> teljesül a nyugalmi teszt: $|f(szülő) - f(csúcs)| < K$,

Példa



min mélység=3
max mélység=5
nyugalmi konstans= 3

Szelektív kiértékelés

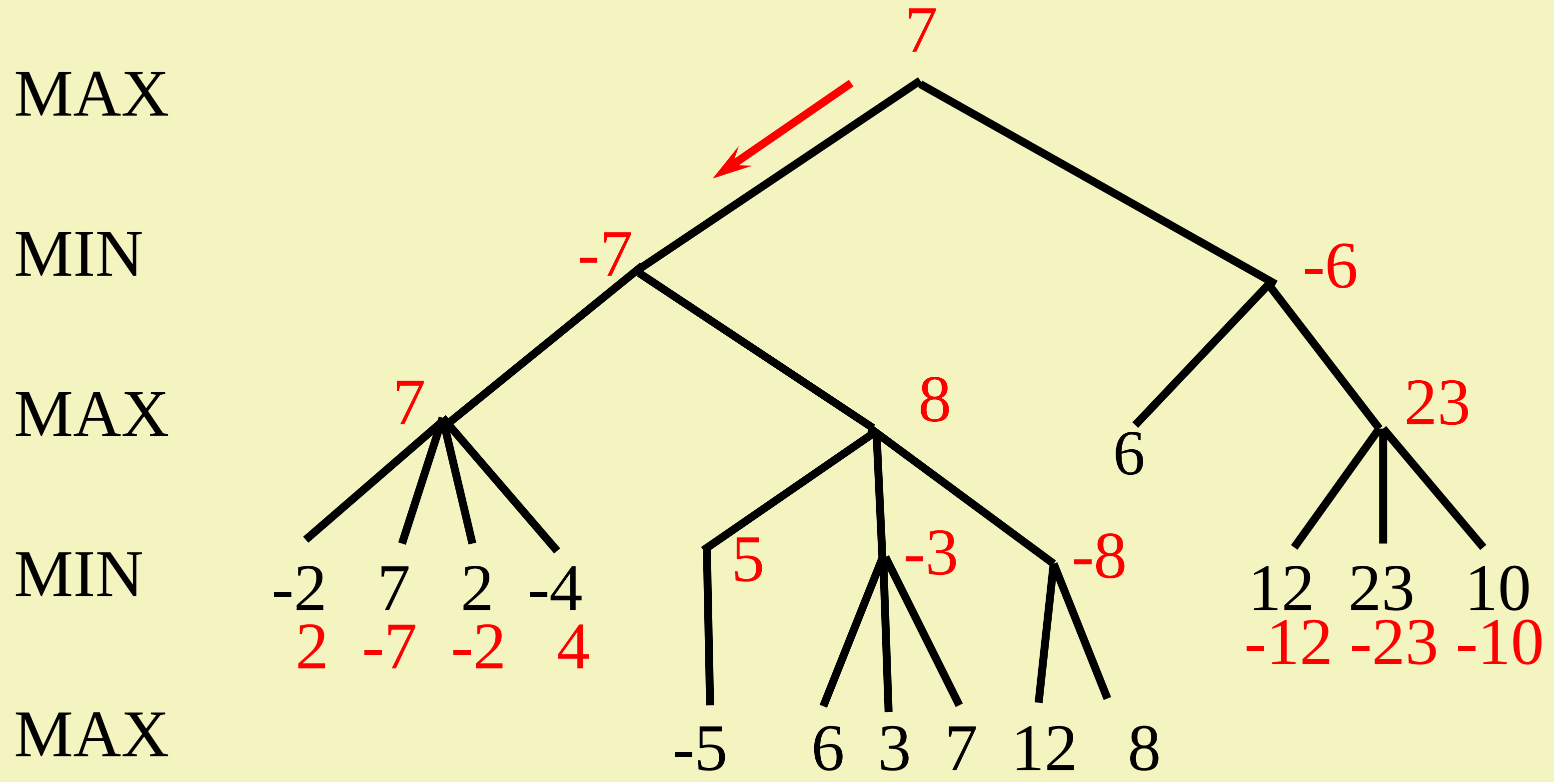
- Célja a memória-igény csökkentése.
- Elkülönítjük a lényeges és lényegtelen lépéseket, és csak a lényeges lépéseknek megfelelő részfát építjük fel.
- Ez a szétválasztás heurisztikus ismeretekre épül.

Negamax algoritmus

- Negamax eljárást könnyebb implementálni.
  - Kezdetben (–1)-gyel szorozzuk azon levélcsúcsok értékeit, amelyek az ellenfél (MIN) szintjein vannak, majd
  - Az értékek felfuttatásánál minden szinten az alábbi módon számoljuk a belső csúcsok értékeit:

$$\textit{szülő} := \max(-\textit{gyerek}_1, \ldots, -\textit{gyerek}_k)$$

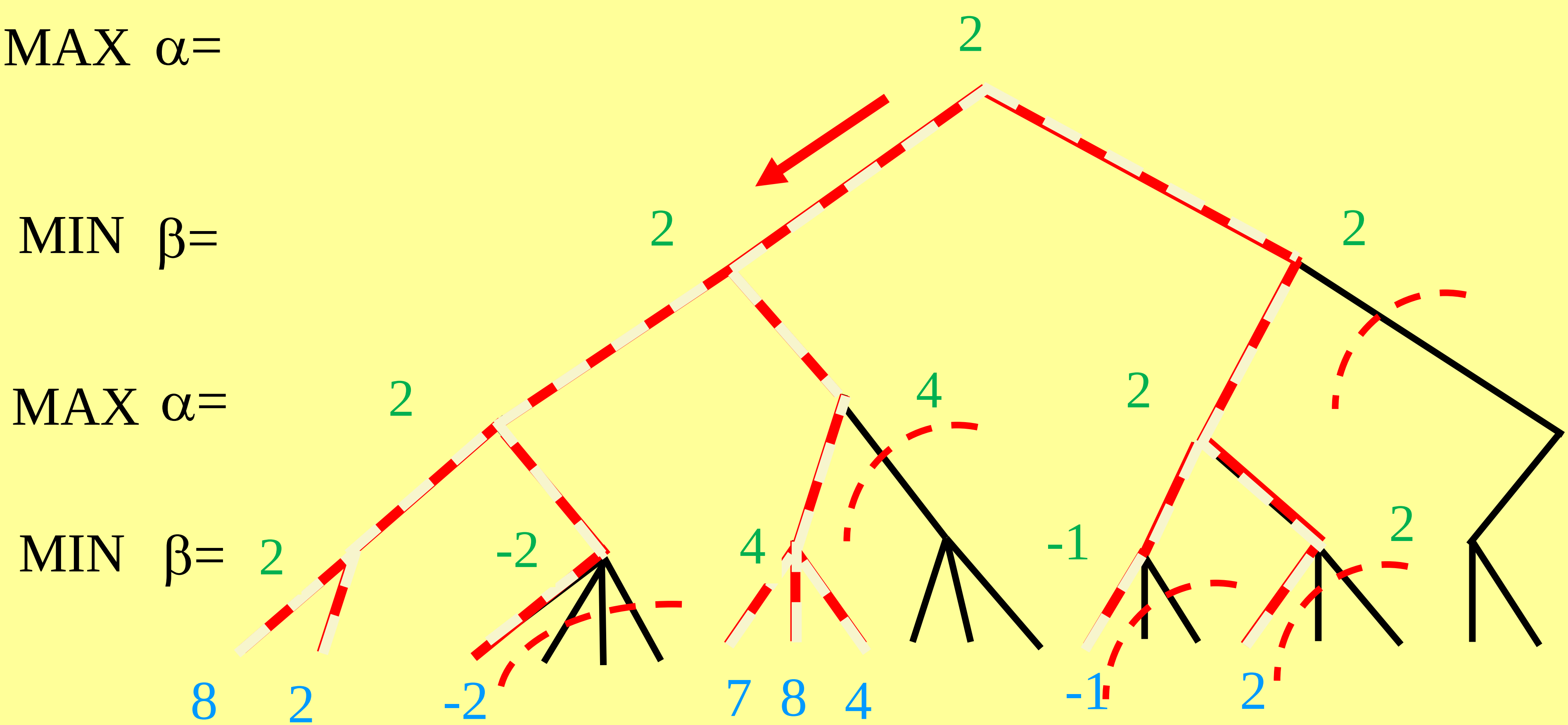
Példa


MAX
MIN
MAX
MIN
MAX
7
-7
-6
7
8
6
23
-2 7 2 -4
2 -7 -2 4
5
-3
-8
12 23 10
-12 -23 -10
-5 6 3 7 12 8

Alfa-béta algoritmus

- Visszalépéses algoritmus segítségével járjuk be a részfát (olyan mélységi bejárás, amely mindig csak egy utat tárol). Az aktuális úton fekvő csúcsok ideiglenes értékei:
  - a MAX szintjein α érték: ennél rosszabb értékű állásba innen már nem juthatunk
  - A MIN szintjein β érték: ennél jobb értékű állásba onnan már nem juthatunk
- Lefelé haladva a fában $\alpha := -\infty$, és $\beta := +\infty$.
- Visszalépéskor az éppen elhagyott (gyermek) csúcs értéke (felhozott érték) módosíthatja a szülő csúcs értékét:
  - a MAX szintjein: $\alpha :=$ *max(felhozott érték,* α*)*
  - a MIN szintjein: $\beta :=$ *min(felhozott érték,* β*)*
- Vágás: ha az úton van olyan α és β, hogy $\alpha \geq \beta$.

Példa



Elemzés

- Ugyanazt a kezdőlépést kapjuk eredményül, amit a minimax algoritmus talál. (Több egyforma kezdőirány esetén a „baloldalit" választjuk.)
- Memória igény: csak egy utat tárol.
- Futási idő: a vágások miatt sokkal jobb, mint a minimax módszeré.
  – Átlagos eset: egy csúcs alatt, két belőle kiinduló ág megvizsgálása után már vághatunk.
  – Optimális eset: egy *d* mélységű *b* elágazású fában kiértékelt levélcsúcsok száma: $\sqrt{b^d}$
  – Jó eset: A részfa megfelelő rendezésével érhető el.

Kétszemélyes játékot játszó program

- Váltakozó mélységű, szelektív, (*m*,*n*) átlagoló, negamax alfa-béta kiértékelést végez.
- Keretprogram, amely váltakozva fogadja a felhasználó lépéseit, és generálja a számítógép lépéseit.
- Kiegészítő funkciók (beállítások, útmutató, segítség, korábbi lépések tárolása, mentés stb.)
- Felhasználói felület, grafika
- Heurisztika megválasztása (kiértékelő függvény, szelekció, kiértékelés sorrendje)