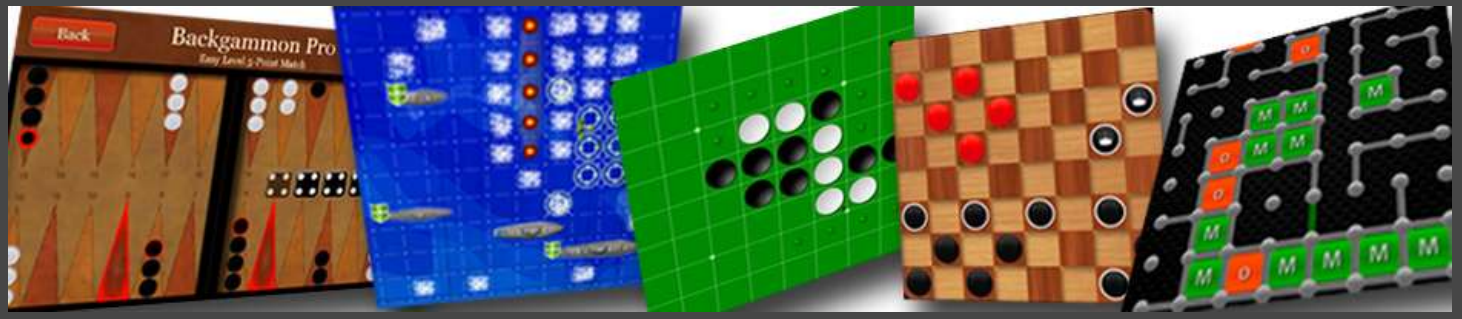
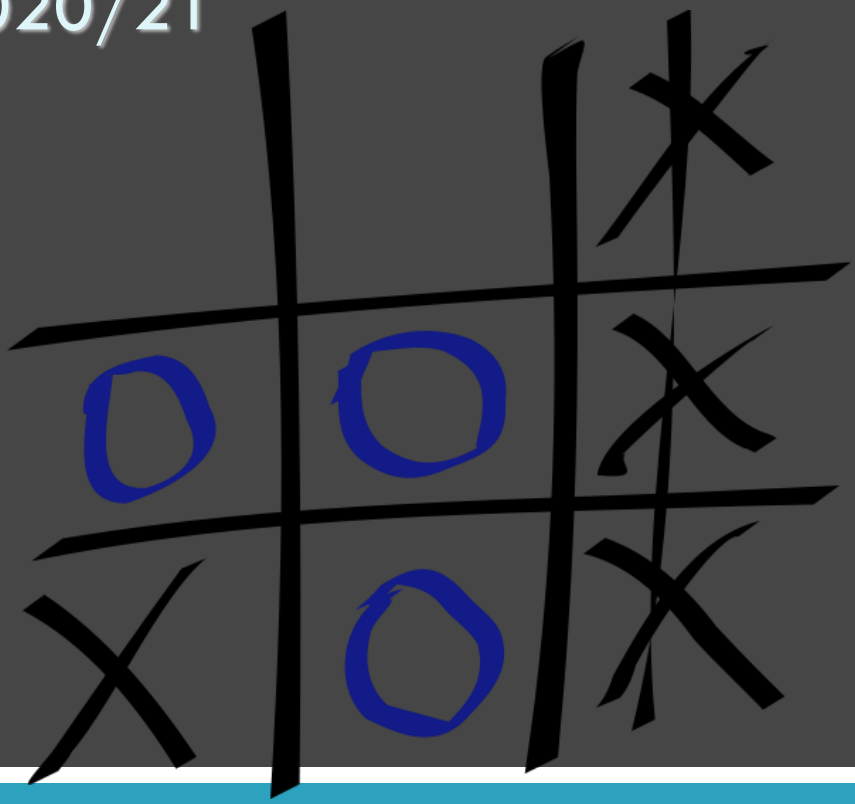


VEŠTAČKA INTELIGENCIJA 2020/21

TRAŽENJE I IGRE

Sadržaj

- MinMax algoritam
- Alfa-Beta odsecanje

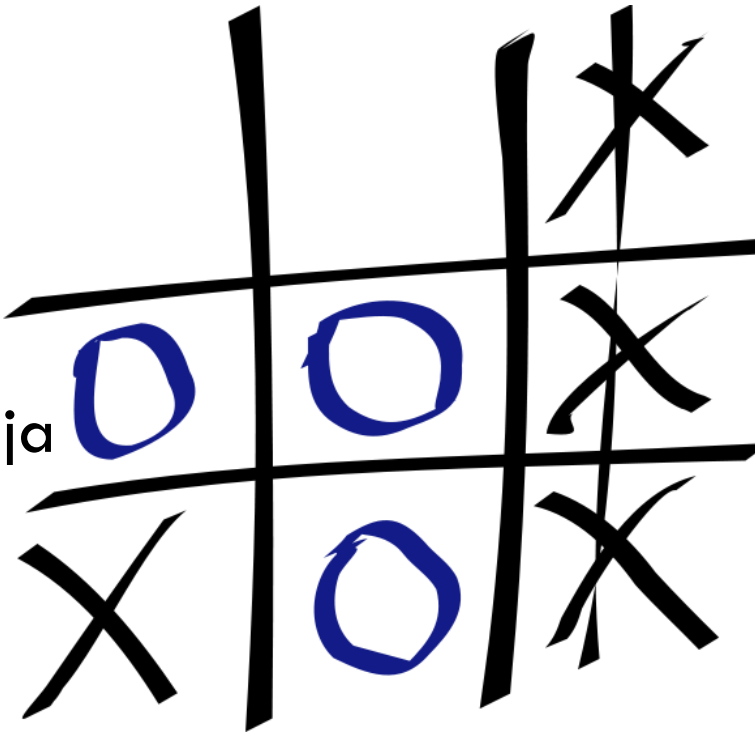


Prof. dr Leonid Stoimenov

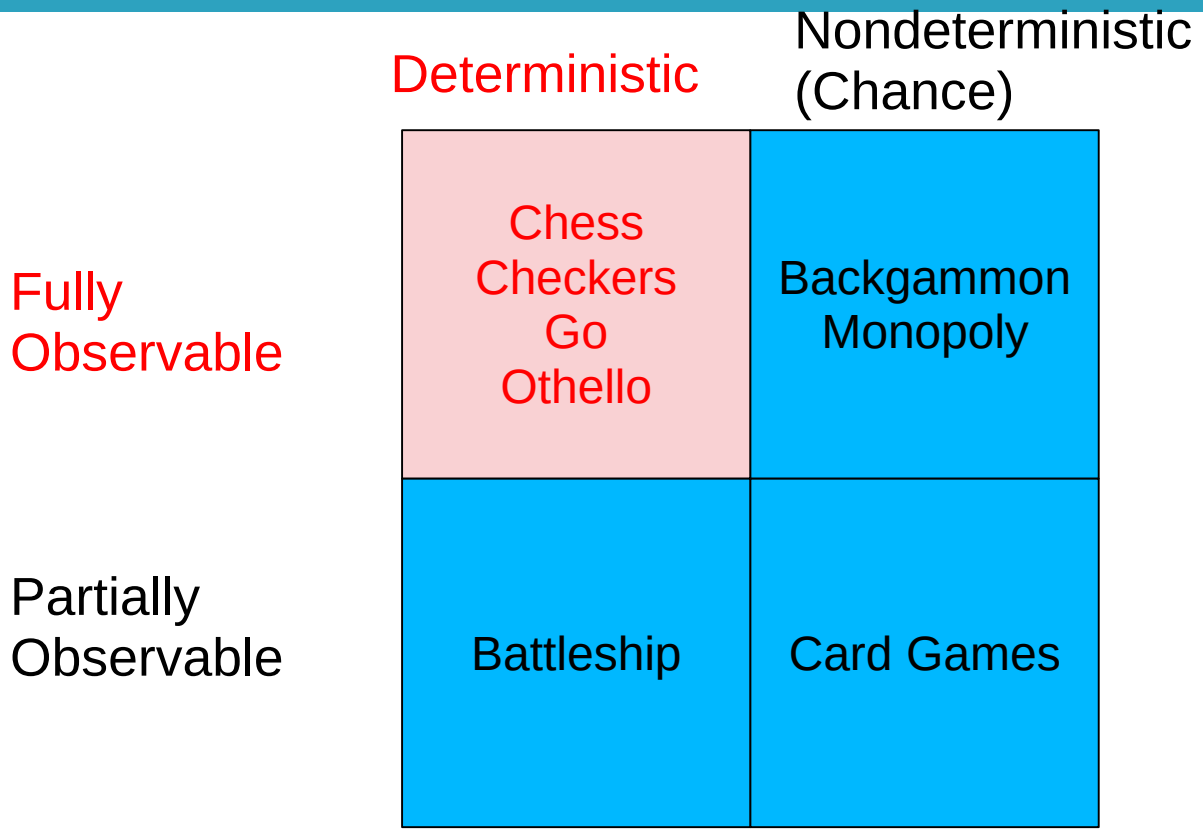
Motivacija

- Kreiranje programa koji igraju igre (logičkog tipa)
- Treba da igra optimalno
- Treba da daje odgovore u razumnom vremenskom periodu

- Razmatramo:
  - MinMax algoritam
  - MinMax sa ograničavanjem traženja
    - Alfa-Beta odsecanje



Tipovi igara



Minimax je za igre bez slučajnih elemenata, samo jedan je pobednik

Traženje i Algoritmi za Igre
Šah, 8-puzzle, tic-tac-toe, mankala,...

- Dva igraca (MIN I MAX) naizmenično povlace poteze
  - *Max* uvek igra prvi.
  - *Min* je protivnik.

Definisanje preko traženja:

- Početno stanje
- Skup operatora
- Test ciljnog stanja (pobeda ili poraz)
- Funkcija korisnosti tj evaluacije (utility function)

Funkcije korisnosti

Max Vs Min

Funkcija korisnosti

Koristi se kao heuristička funkcija s tom razlikom što se preko nje vrši **evaluacija čvora** (koji se odnosi na potez nekog igrača) u smislu **procene** koliko je on **koristan za svakog od igrača**

Pozitivne vrednosti indiciraju stanja koja obezbeđuju **prednost za Max**

Negativne vrednosti indiciraju stanja koja obezbeđuju **prednost za Min**

Funkcija korisnosti (evaluacije)

- Procenjuju vrednost stanja na osnovu njegovih osobina – daje korisnost nekog stanja igre (utility(State): u(s))
- Primer: -1, 0, +1:
  - Igrač 1 gubi,
  - nerešeno,
  - Igrač 1 dobija (respektivno)

+1, -1, 0 za mnoge igre

IKS-OKS: Primer funkcije korisnosti

Ako Max igra kao “X”

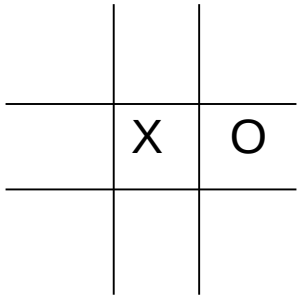
$U(n)$ =

if n je pobeda za Max, + ∞

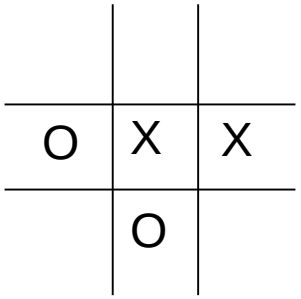
if n je pobeda za Min, - ∞

else

(broj vrsta, kolona i dijagonala dostupnih za Max) - (broj vrsta, kolona i dijagonala dostupnih za Min)



$U(n) = 6 - 4 = 2$



$U(n) = 4 - 3 = 1$

Šah: Primer funkcije korisnosti

Ako je Max: “White”

Pretpostavka da svaka figura ima sledeće vrednosti

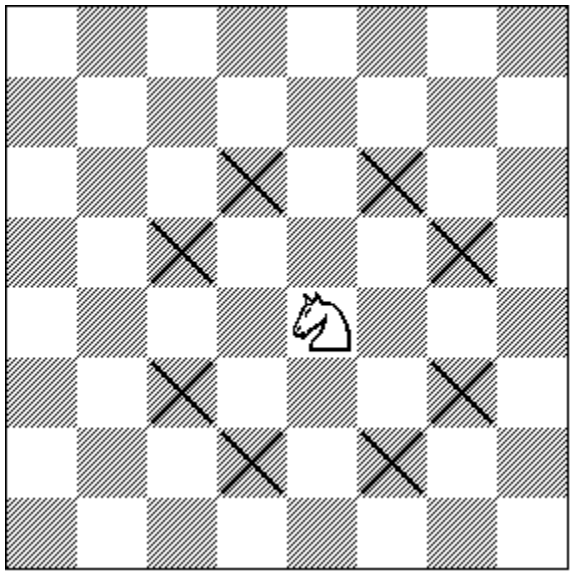
pion = 1;
Kralj = 3;
Lovac = 3;
Top = 5;
Kraljica = 9;

w = suma vrednosti belih figura
b = suma vrednosti belih figura

$$U(n) = \frac{w - b}{w + b}$$

Vrednost ide između 1 i -1

Ova funkcija naivno daje istu težinu figurama bez obzira na poziciju na tabli...
Kako doći do dobre funkcije?
- Intervjuisati eksperta
- Mašinsko učenje



Šah: Primer funkcije korisnosti

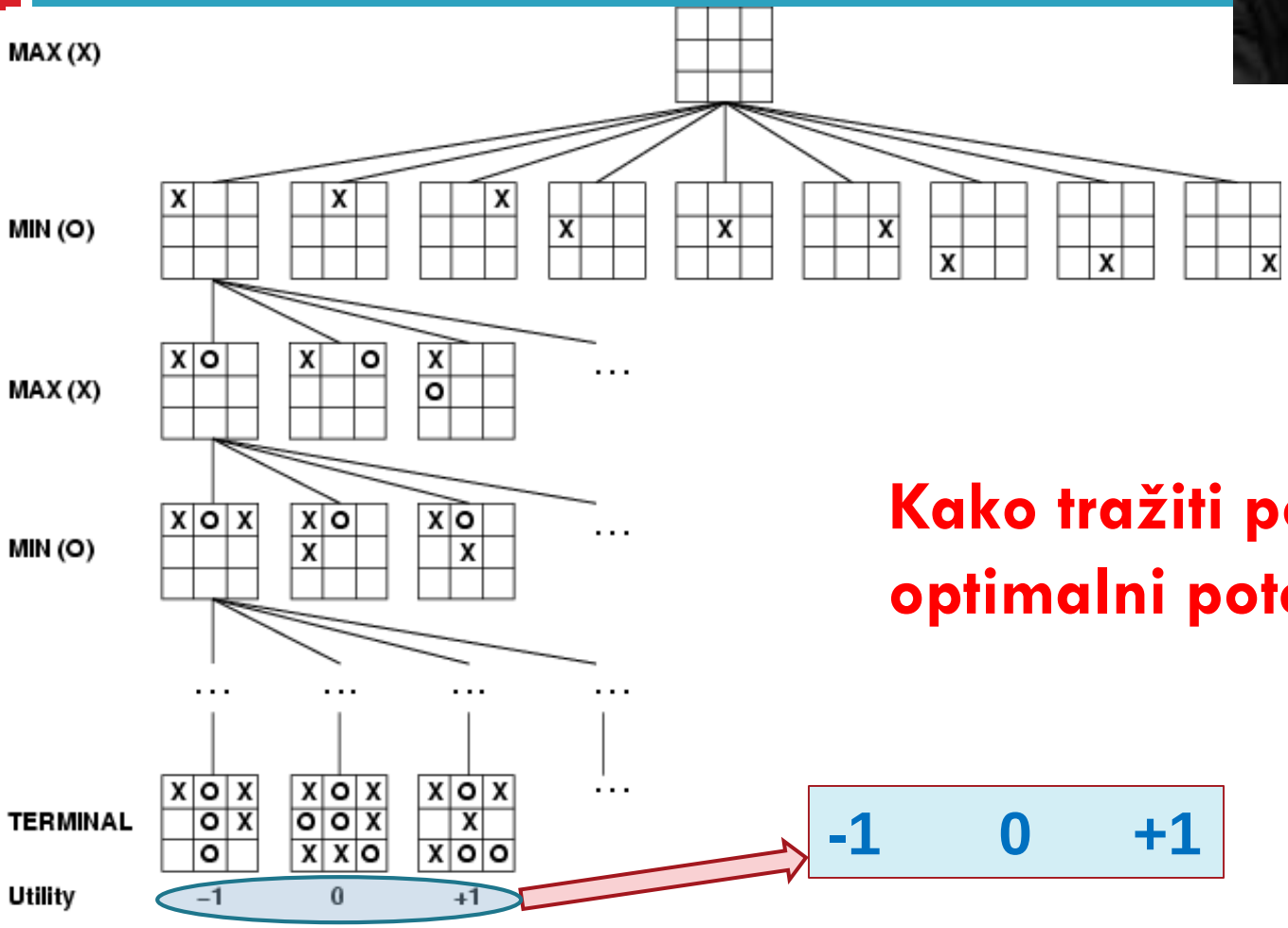
- Linearna težinska funkcija
- $utility(S) = w_1 f_1(S) + w_2 f_2(S) + \ldots + w_n f_n(S)$
  - $f_1(S)$ = (Number of white queens) – (Number of black queens), $w_1 = 9$
  - $f_2(S)$ = (Number of white rooks) – (Number of black rooks), $w_2 = 5$
  - ...

Minimax algoritam – osnovna ideja

- Generiše **kompletno stablo traženja**
  - Na svakom nivou jedan od igraca povlači potez
  - Traženje po stablu mogućih poteza sa ciljem da se nađe potez koji prouzrokuje najbolji rezultat
- Analiza celokupnog stabla daje optimalne poteze (za oba igrača – pretpostavka da i protivnik igra optimalno!)
- **Depth First Search** (DFS) algoritam
- Generalno nije izvodljivo – stablo može biti preveliko!

Stablo traženja za igru (2-igrača, deterministička)

Max Vs Min



Kako tražiti po stablu za optimalni potez?

-1 0 +1

Postavke za Minimax

- Inicijalno stanje
  - Pozicija tj stanje na tabli
  - Ko je na potezu
- Operatori
  - Legalni potezi igrača
- Test
  - Određuje da li je stanje ciljno
- Funkcija korisnosti (zavisi od igre)

Implementacija: Dva agenta

- MAX
  - Pokušava da **maksimizuje** rezultat funkcije korisnosti
  - Pobednička strategija, ako, na MIN potez po redu, pobeda je dostižna za MAX za sve moguće poteze MIN
- MIN
  - Pokušava da **minimizuje** rezultat funkcije korisnosti
  - Pobednička strategija, ako, na MAX potez po redu, pobeda je dostižna za MIN, za sve moguće poteze MAX

MinMax algoritam

```
function MINIMAX-DECISION(state) returns an action
   inputs: state, current state in game
   return the a in ACTIONS(state) maximizing MIN-VALUE(RESULT(a, state))

function MAX-VALUE(state) returns a utility value
   if TERMINAL-TEST(state) then return UTILITY(state)
   v ← −∞
   for a, s in SUCCESSORS(state) do v ← MAX(v, MIN-VALUE(s))
   return v

function MIN-VALUE(state) returns a utility value
   if TERMINAL-TEST(state) then return UTILITY(state)
   v ← ∞
   for a, s in SUCCESSORS(state) do v ← MIN(v, MAX-VALUE(s))
   return v
```

MinMax algoritam

Maks i Mini igraju igru

U(list_stabla) = P**(list_stabla)**

1 ako Maks pobedi

0 za nerešeno

-1 ako Maks gubi, tj Mini dobija

Čvor X:

U(X) =

ako je X list stabla

P (x)

inače

max {U(c) | c je potomak X}

ako Maks igra sledeći

min {U(c) | c je potomak X}

ako Mini igra sledeća

Primer: Stablo traženja

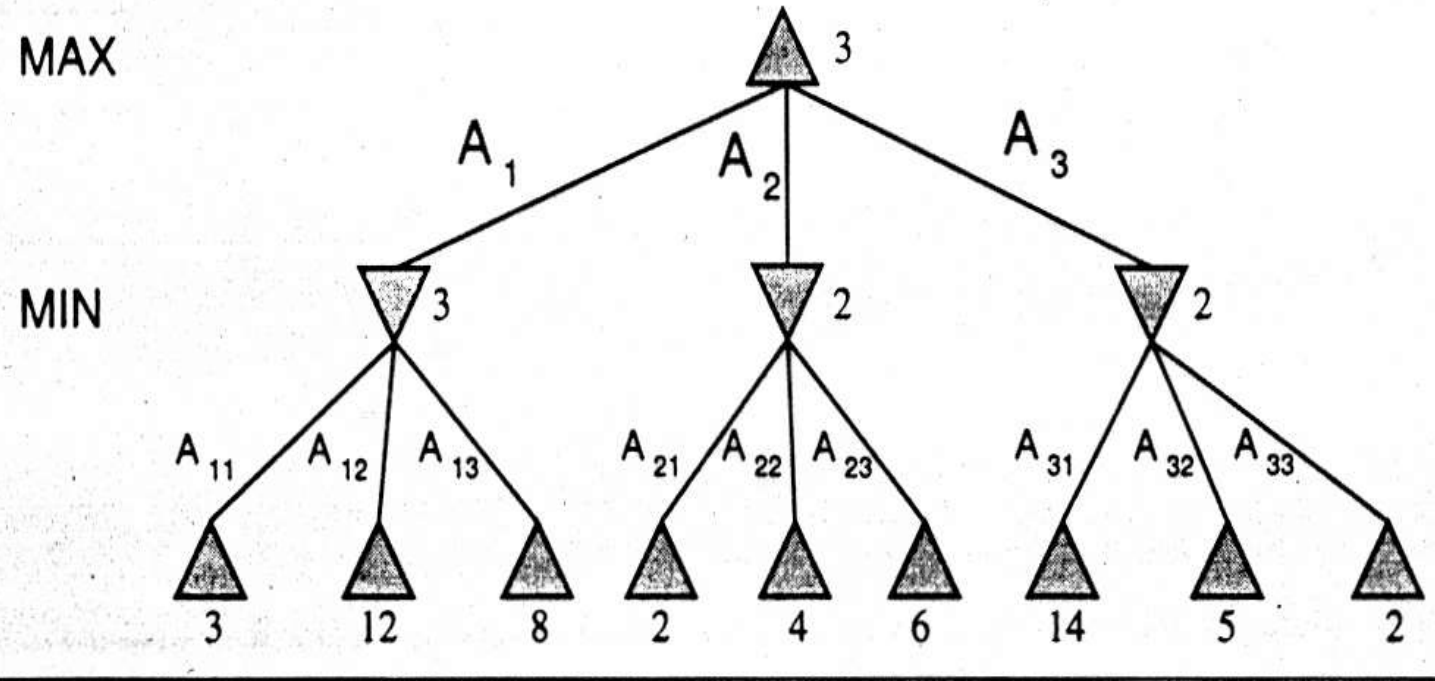
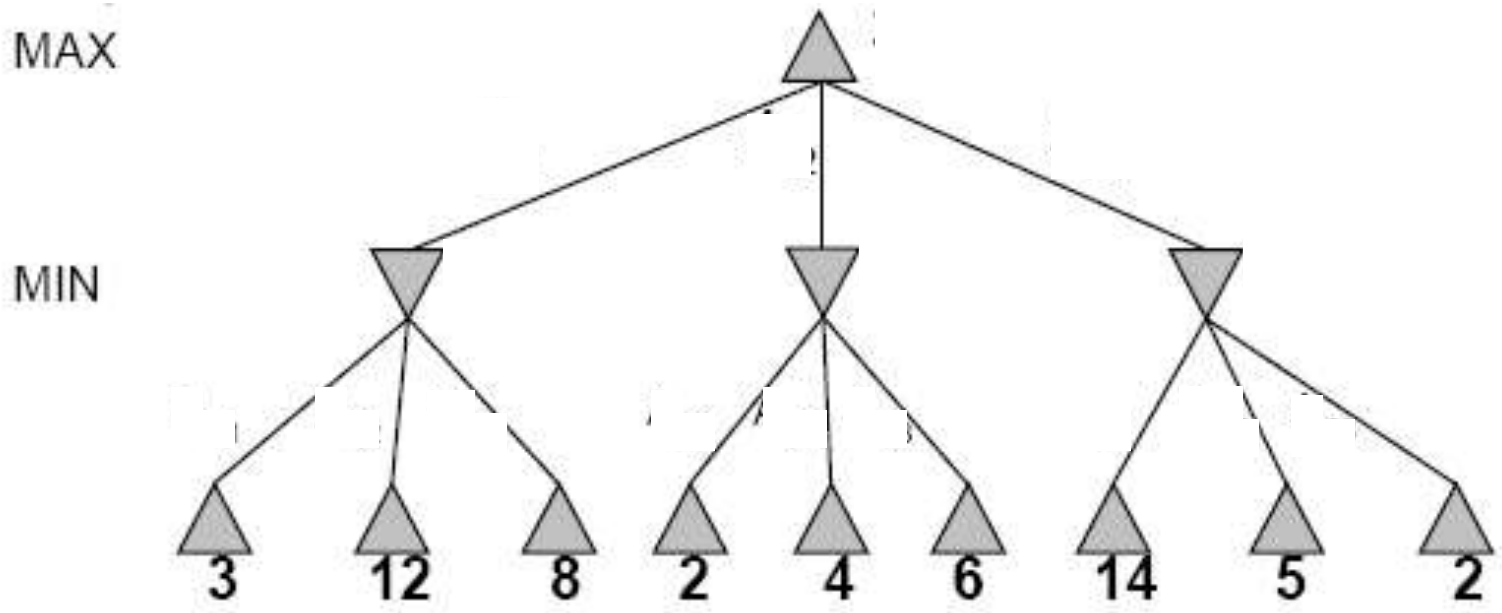
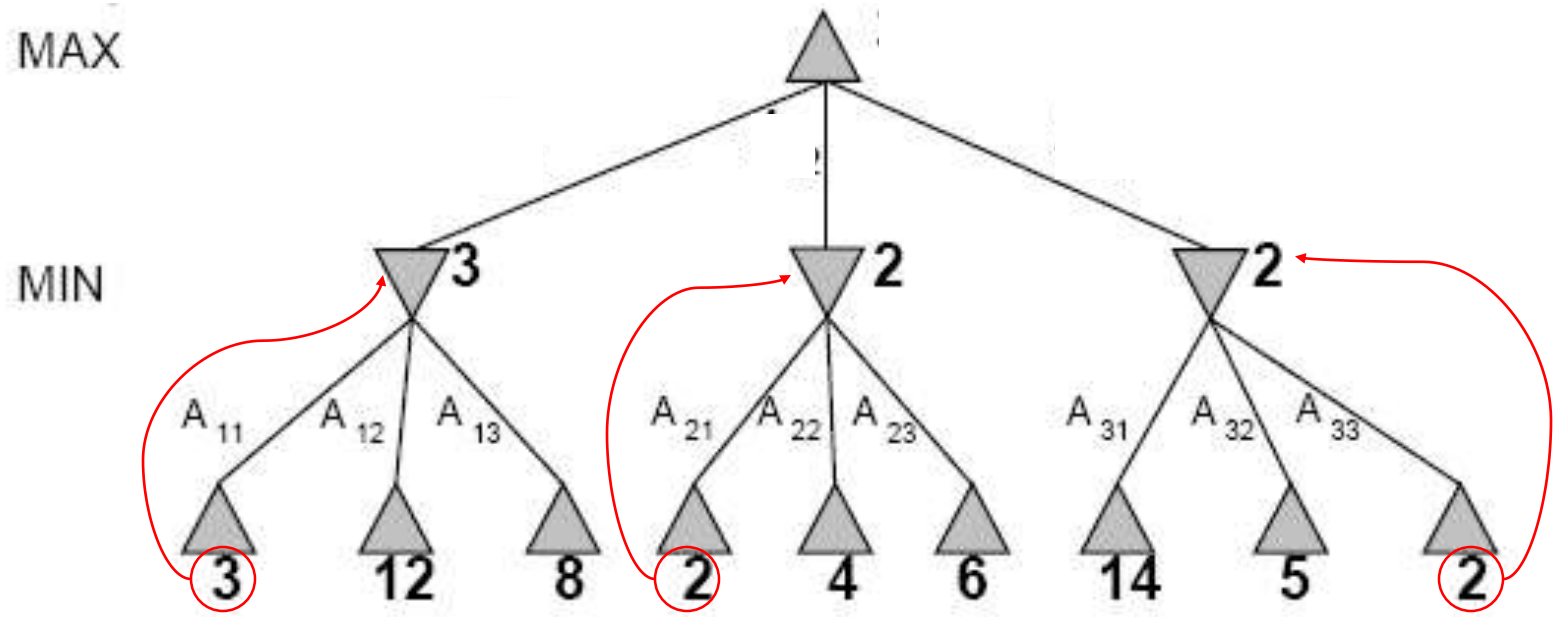


Figure 5.2 A two-ply game tree as generated by the minimax algorithm. The △ nodes are moves by MAX and the ▽ nodes are moves by MIN. The terminal nodes show the utility value for MAX computed by the utility function (i.e., by the rules of the game), whereas the utilities of the other nodes are computed by the minimax algorithm from the utilities of their successors. MAX's best move is A_1, and MIN's best reply is A_{11}.

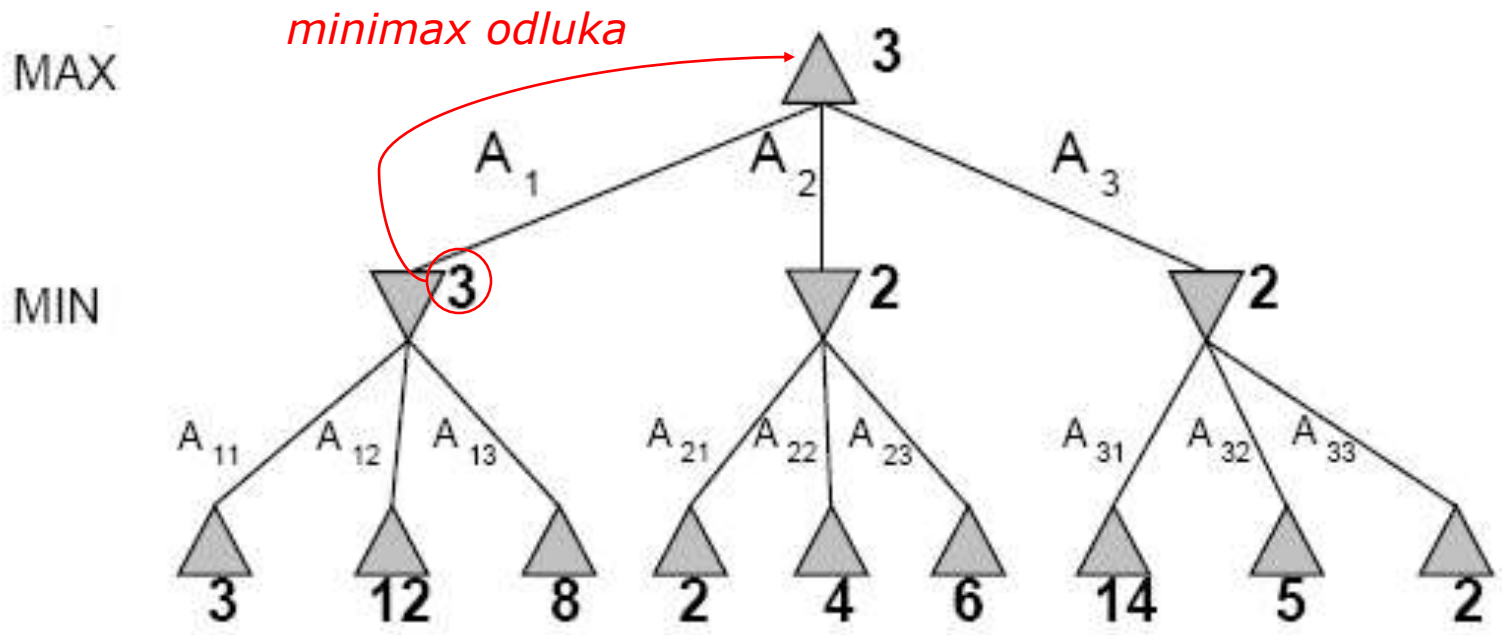
Primer rada Min-Max



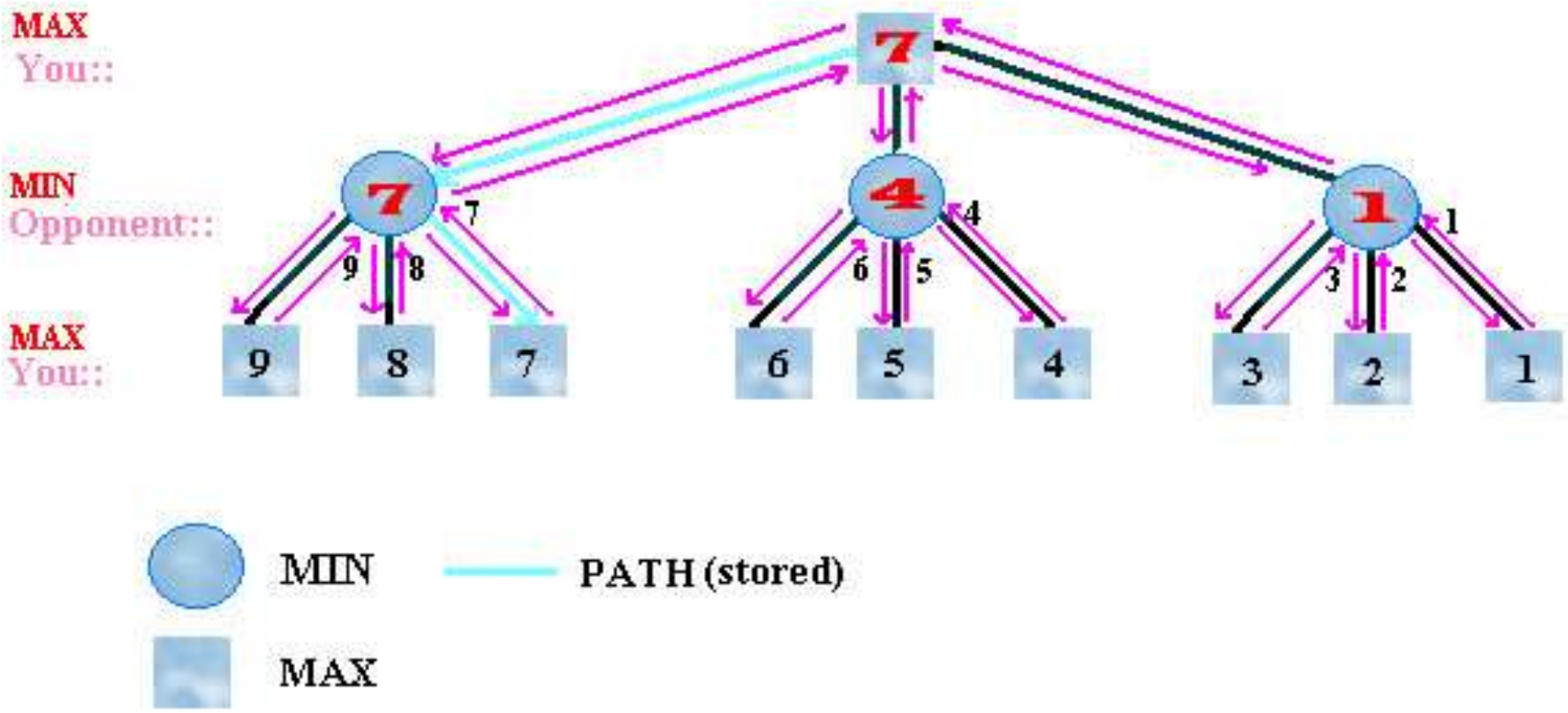
Primer rada Min-Max



Primer rada Min-Max



MinMax traženje

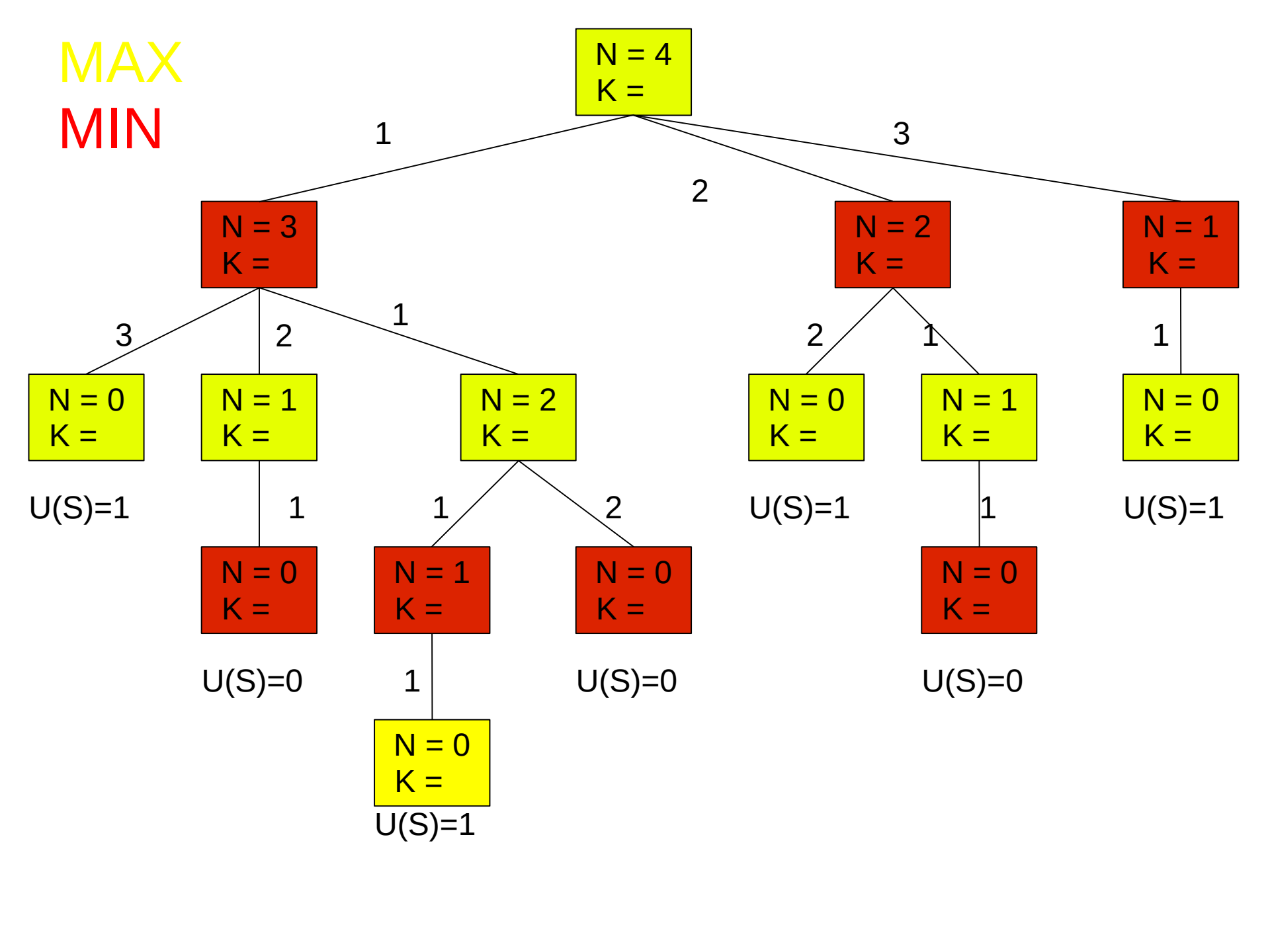


Primer - *Coins game*

- Magacin sa N kovanica
- Kad je na potezu, igrač uzima 1, 2, ili 3 kovanice iz magacina
- Igrač koji uzme poslednju kovanicu gubi

Coins Game: Formalna definicija

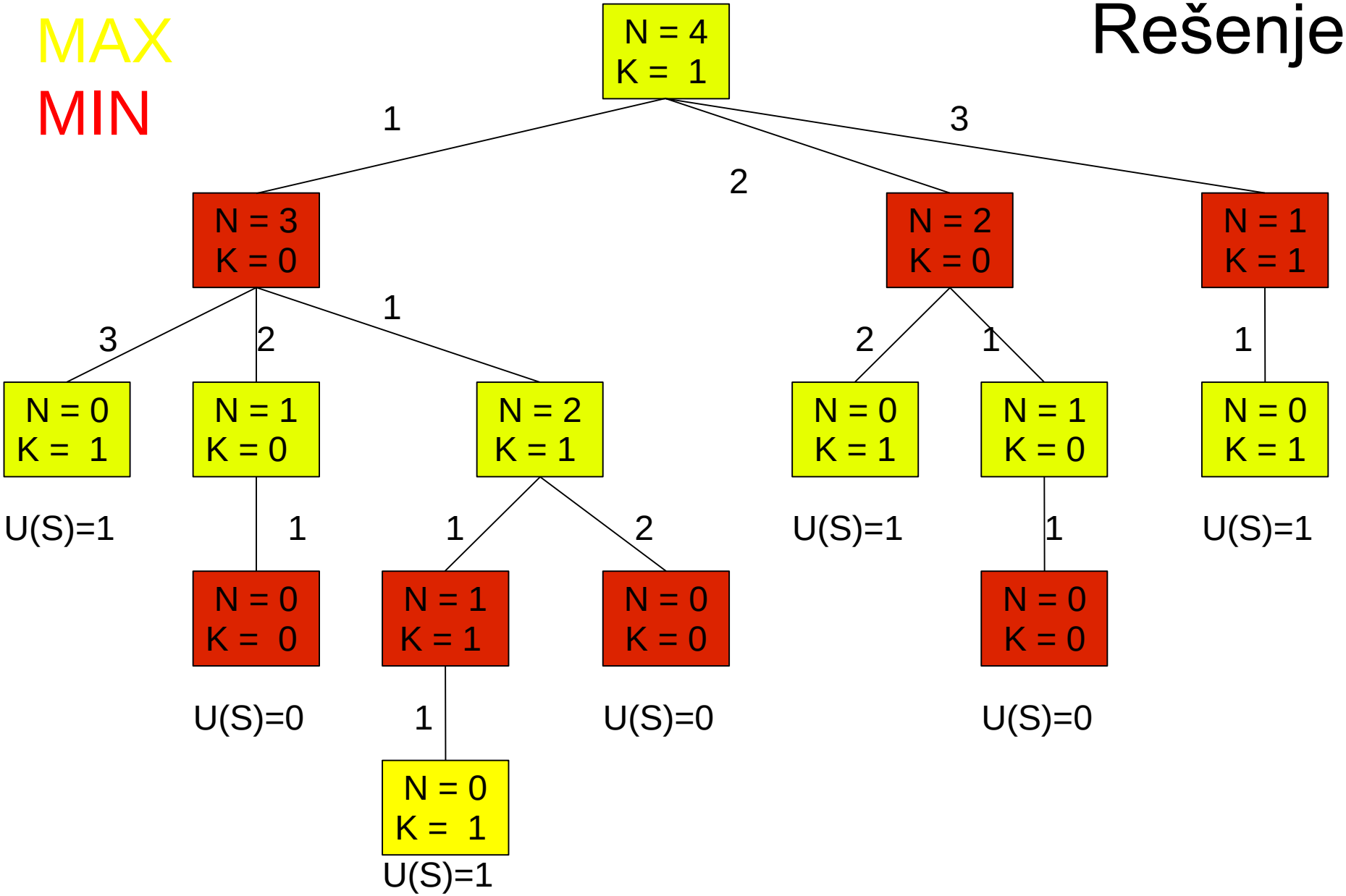
- Početno stanje: Broj kovanica u magacinu
- Operatori:
  1. Uzmi jednu kovanicu
  2. Uzmi dve kovanice
  3. Uzmi tri kovanice
- Test: Nema kovanica u magacinu
- Utility Function: U(S)
  - U(S) = 1 ako MAX pobedi, 0 ako MIN pobedi


MAX
MIN
N = 4
K =
1
2
3
N = 3
K =
N = 2
K =
N = 1
K =
3
2
1
2
1
1
N = 0
K =
N = 1
K =
N = 2
K =
N = 0
K =
N = 1
K =
N = 0
K =
U(S)=1
1
1
2
U(S)=1
1
U(S)=1
N = 0
K =
N = 1
K =
N = 0
K =
N = 0
K =
U(S)=0
1
U(S)=0
U(S)=0
N = 0
K =
U(S)=1

Rešenje

MAX

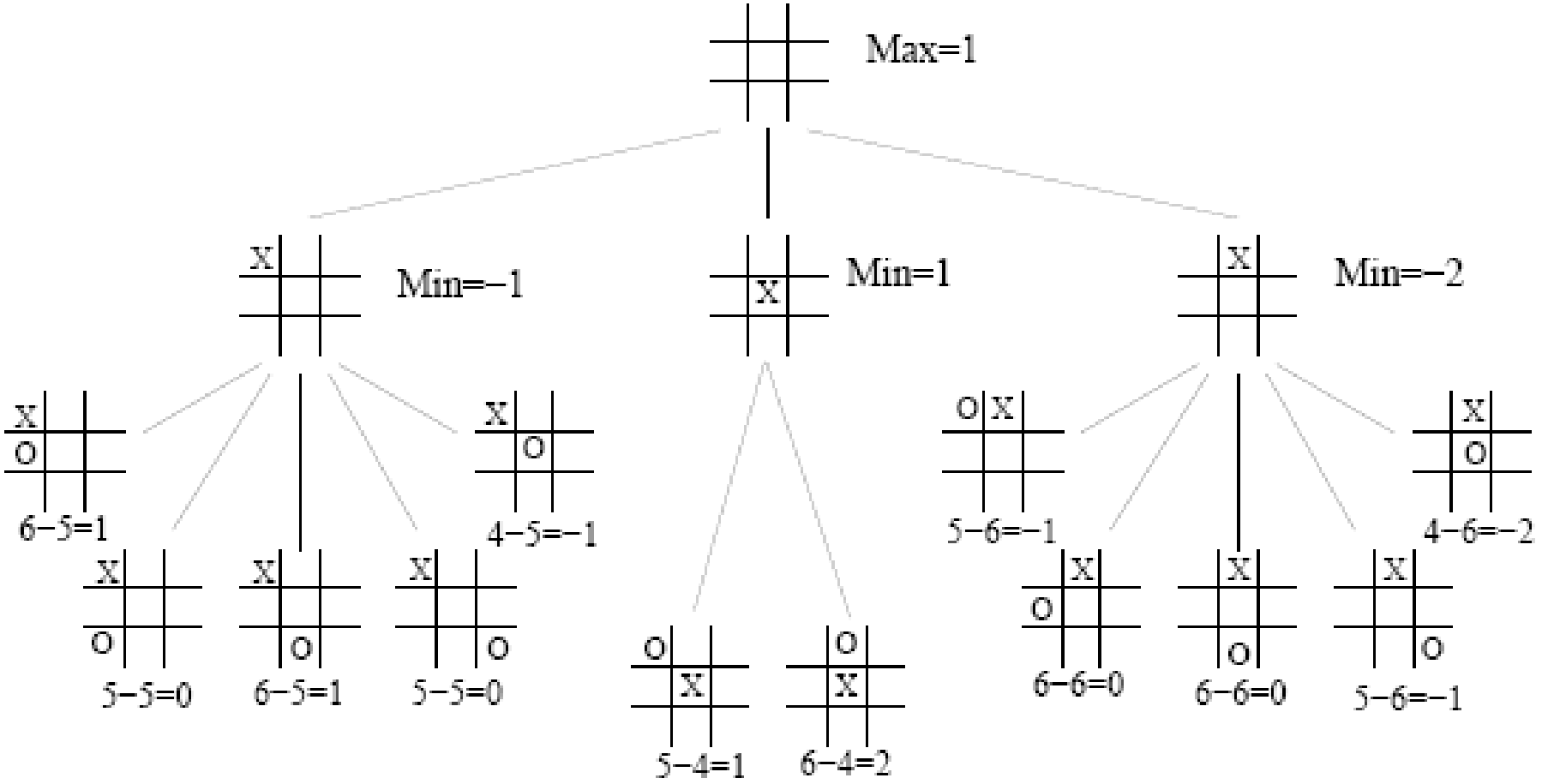
MIN



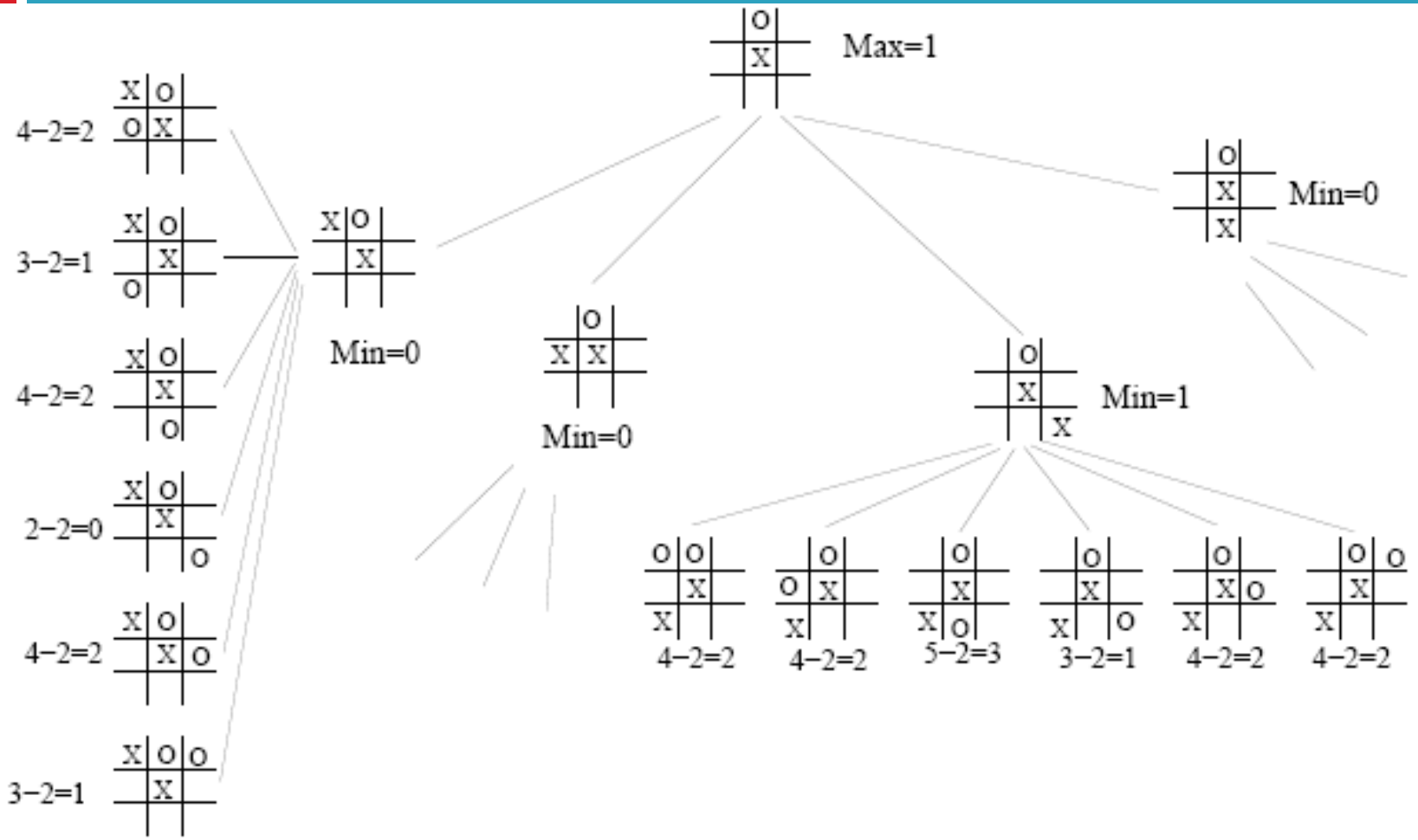
Analiza

- Maks. Dubina (*d*): *5*
- Faktor grananja (*b*): 3
- Broj čvorova: 15
- Iako je trivijalan primer, stablo je poprilično veliko
  - Generalno, ima $O(b^d)$ čvorova za pretragu
  - Vreme izvršenja algoritma raste eksponencijalno!
  - Kako ga ubrzati?
- Ograničiti traženje po dubini
  - Pretražujete sve dok možete (vreme, zadata dubina)
  - Evaluacija stanja i vraćanje do korena

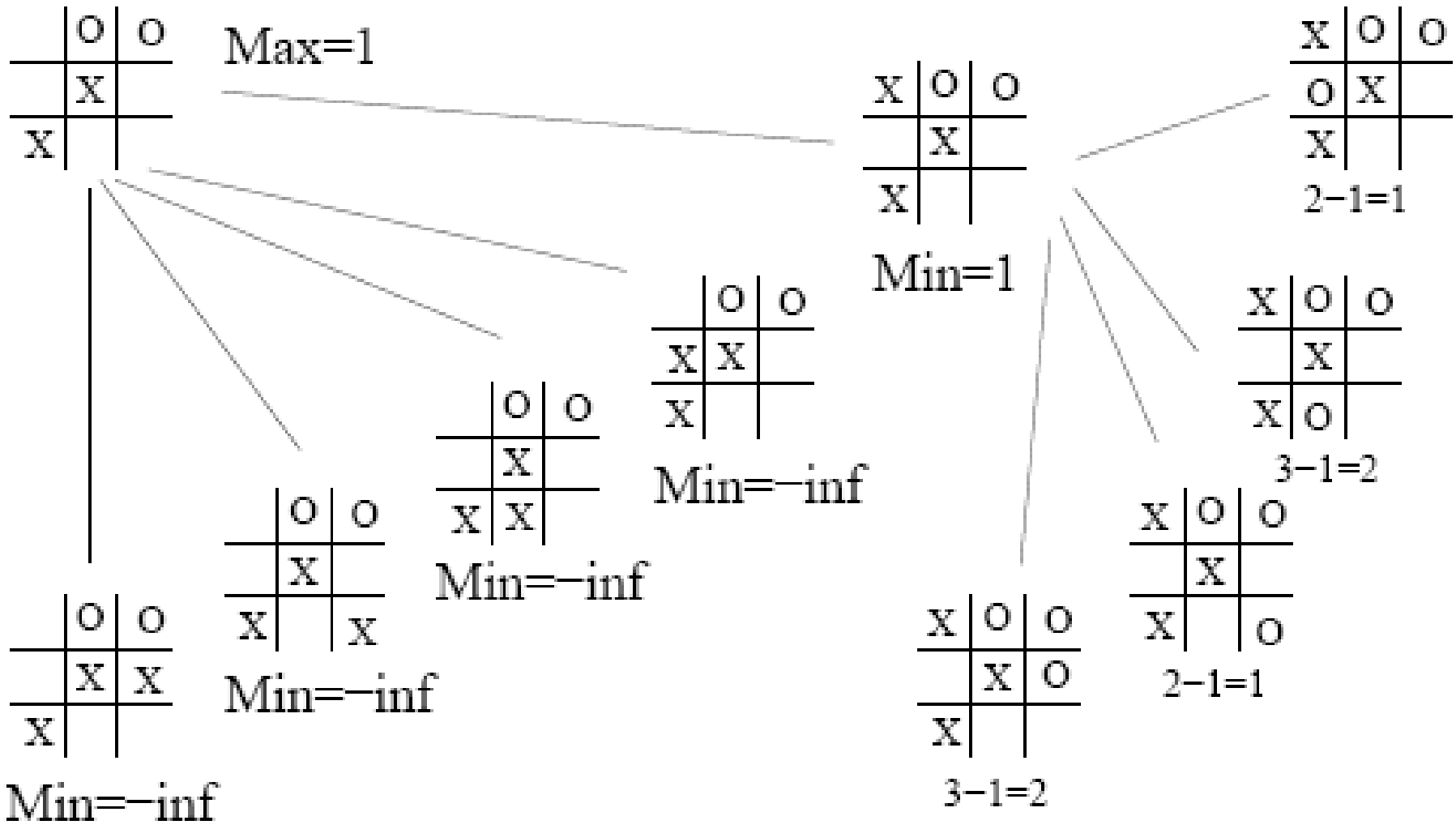
Primer: X-O


Max=1
Min=−1
Min=1
Min=−2
6−5=1
4−5=−1
5−5=0
6−5=1
5−5=0
5−4=1
6−4=2
5−6=−1
4−6=−2
6−6=0
6−6=0
5−6=−1

Primer: X-O


Max=1
Min=0
Min=0
Min=0
Min=1
4−2=2
3−2=1
4−2=2
2−2=0
4−2=2
3−2=1
4−2=2
4−2=2
5−2=3
3−2=1
4−2=2
4−2=2

Primer: X-O



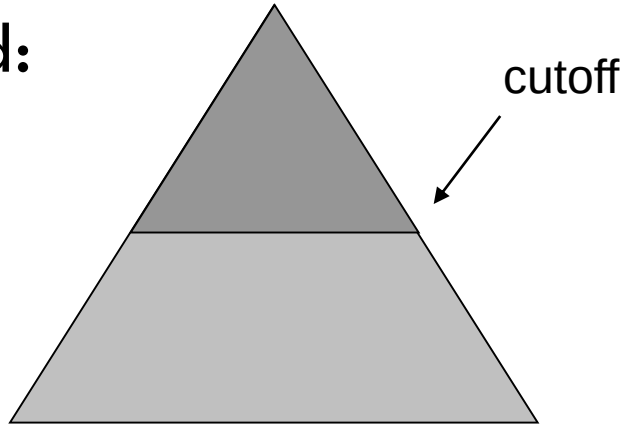
MinMax ograničen po dubini

- Za čvor X koji se nalazi na dubini d:

  U (x, d) =

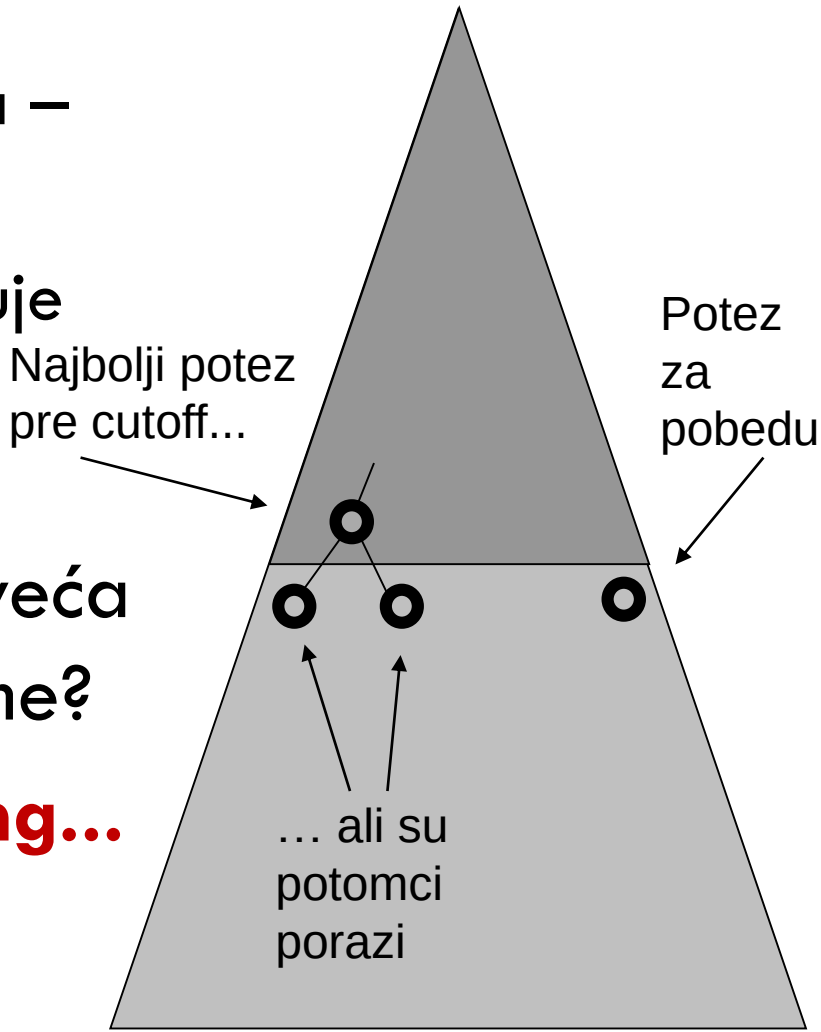
  if (x je list) ILI (d > max. dubina)
  - P (x)

  else
  - max {U (c, d + 1) | c je dete čvora x} ako je potez tog igrača na redu.
  - min {U (c, d + 1) | c c je dete čvora x} ako je potez drugog igrača na redu.



MinMax ograničen po dubini (2)

- MinMax:
  - Zbog vremenskog ograničenja – ograničavanje po dubini
  - Korišćenje *cutoff* –a prouzrokuje probleme zbog tzv efekta "horizont".
- Da li postoji način da se poveća dubina traženja za isto vreme?
- **Rešenje: Alpha-Beta Pruning...**



Alfa-beta odsecanje (α-β pruning)

- Cilj algoritma: **smanjiti stablo traženja**
- Odseca grane koje ne obećavaju
- Osnovna ideja:
  - Izbegavaju se podstabla koja nemaju efekat na rezultat
  - Pamti se vrednost **najboljeg** poteza **do sada**
    - α – najbolja vrednost za igraca Max. Koristi se u MIN čvorovima, i dodeljuje u MAX čvorovima
    - β – najbolja vrednost za igraca Min. Koristi se u MAX čvorovima i dodeljuje u MIN čvorovima
- Kada **Max** ispituje moguće akcije, ako je bilo koja od njih veća od β (sto je gore po Min), onda može da se prestane sa traženjem (podrazumeva se da Min nece odigrati potez koji nije dobar).

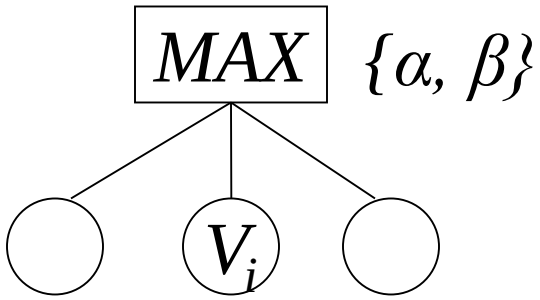
Alpha-Beta odsecanje

- MAX (ne na nivou 0)
  - Ako je nađeno podstablo sa vrednošću ***k*** koja je veća od vrednosti β, nema potrebe za njegovom daljom pretragom
    - MAX može da ima potez koji je dobar najmanje ***k***, tako da MIN nikad neće da izabere taj put!
- MIN
  - Ako je nađeno podstablo sa vrednošću ***k*** koja je manja od vrednosti α, nema potrebe za njegovom daljom pretragom
    - MIN može da ima potez koji je dobar najmanje ***k***, tako da MAX nikad neće da izabere taj put!

Algoritam sa Alfa-beta odsecanjem

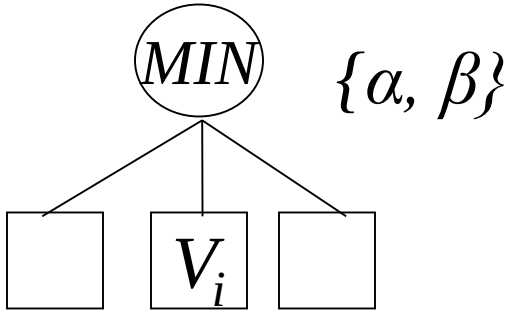
```
function ALPHA-BETA-DECISION(state) returns an action
  return the a in ACTIONS(state) maximizing MIN-VALUE(RESULT(a, state))

function MAX-VALUE(state, α, β) returns a utility value
  inputs: state, current state in game
          α, the value of the best alternative for MAX along the path to state
          β, the value of the best alternative for MIN along the path to state
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← −∞
  for a, s in SUCCESSORS(state) do
    v ← MAX(v, MIN-VALUE(s, α, β))
    if v ≥ β then return v
    α ← MAX(α, v)
  return v

function MIN-VALUE(state, α, β) returns a utility value
  same as MAX-VALUE but with roles of α, β reversed
```

Alpha-beta odsecanje



If $V_i > \alpha$, modify α
If $Vi \geq \beta$, β pruning

Return α

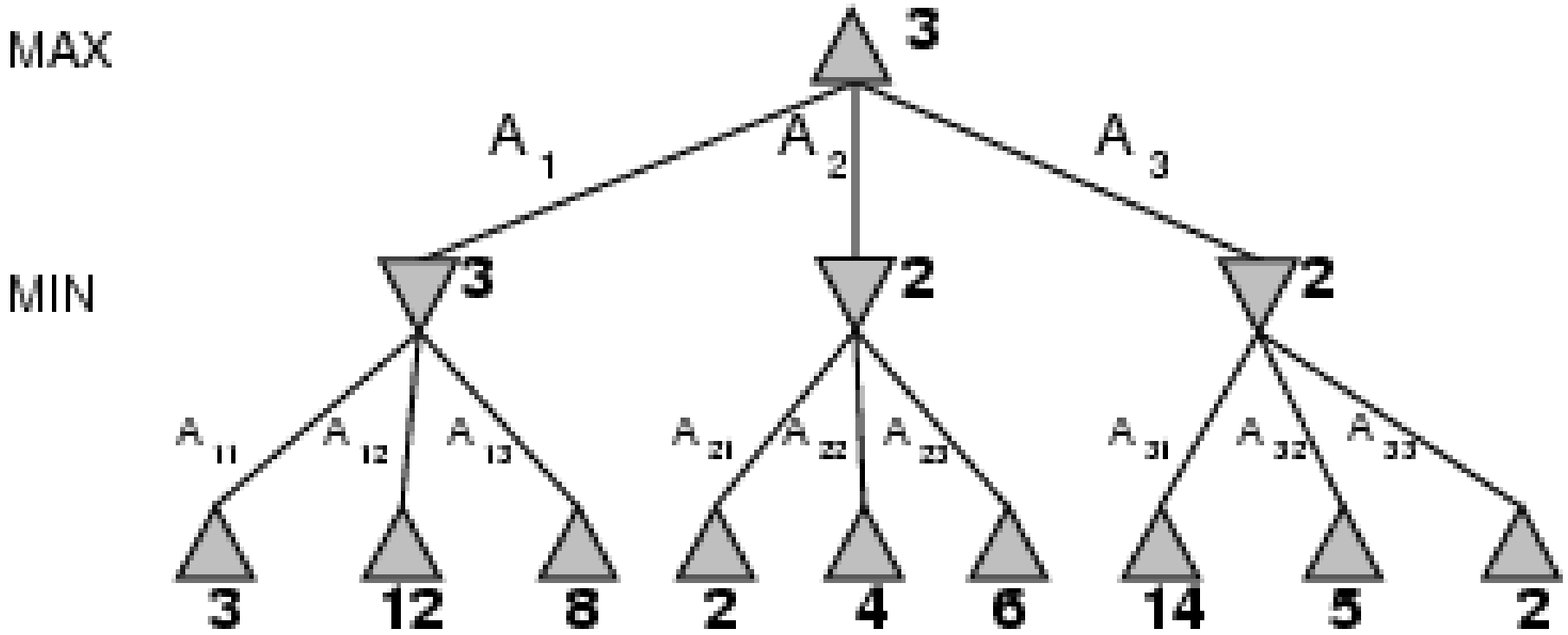


If $V_i < \beta$, modify β
If $Vi \leq \alpha$, α pruning

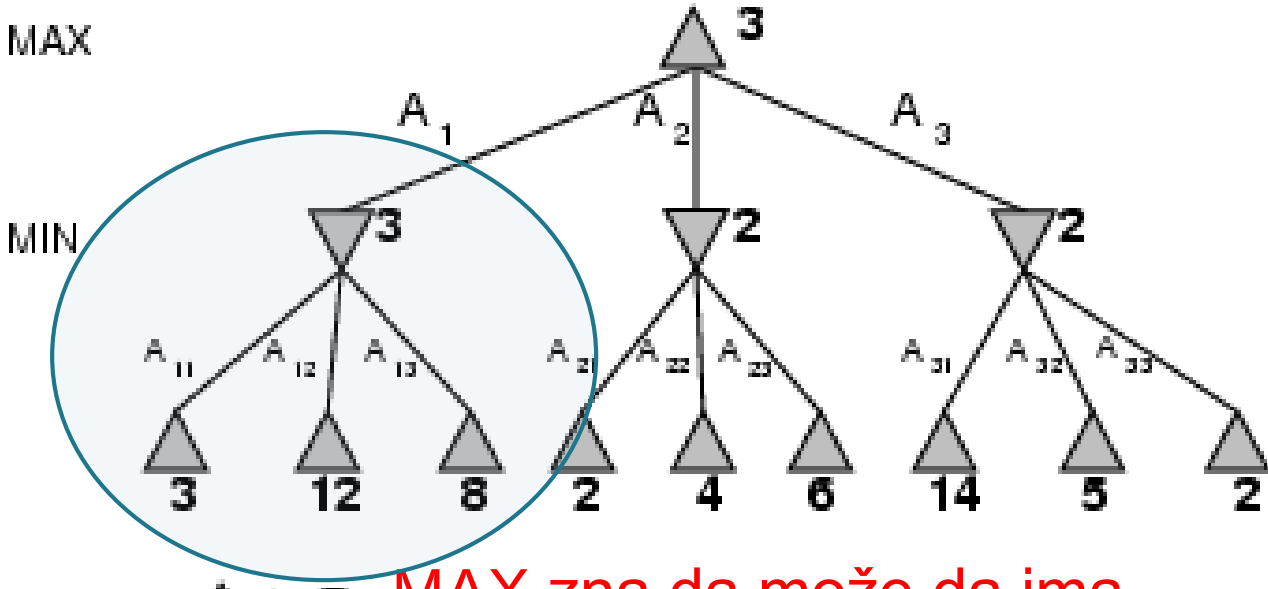
Return β

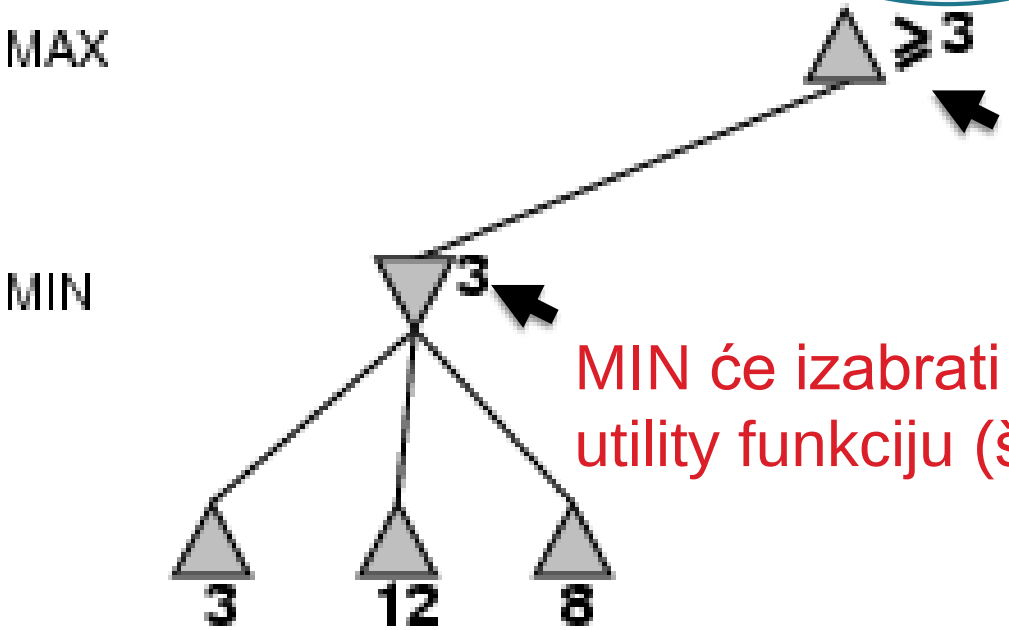
Ilustracija rada: α-β pruning

Kompletno min-max stablo



Ilustracija rada: α-β pruning

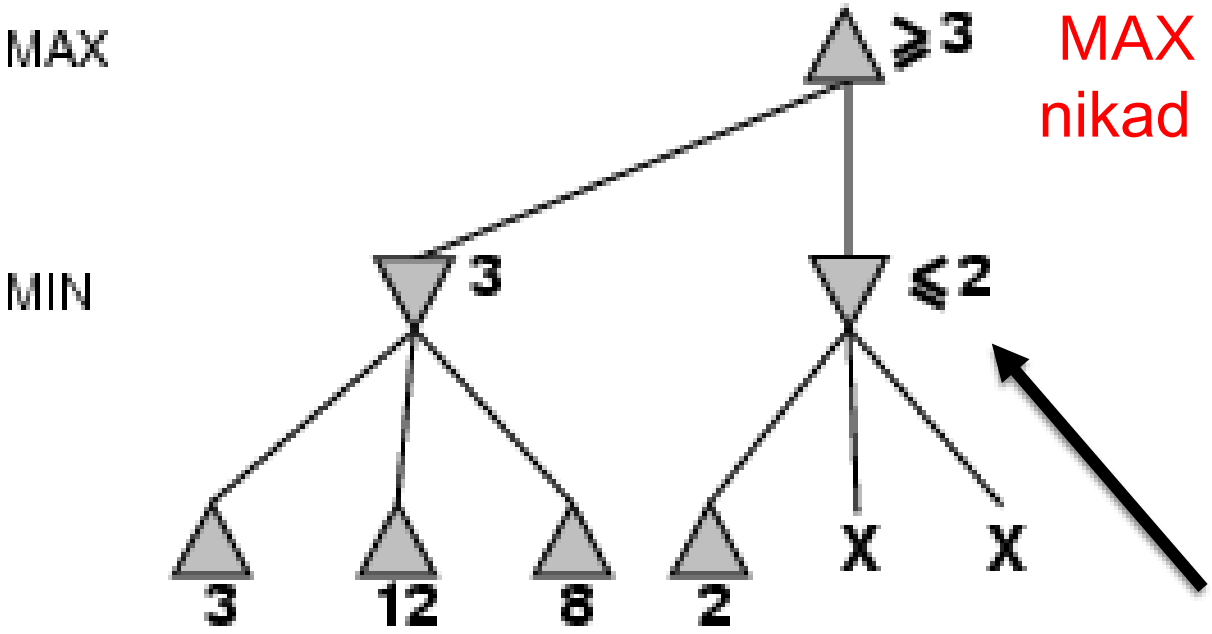




MAX zna da može da ima najmanje “3” ako odigra potez u ovoj grani

MIN će izabrati “3”, zato što minimizira utility funkciju (što je dobro za MIN)

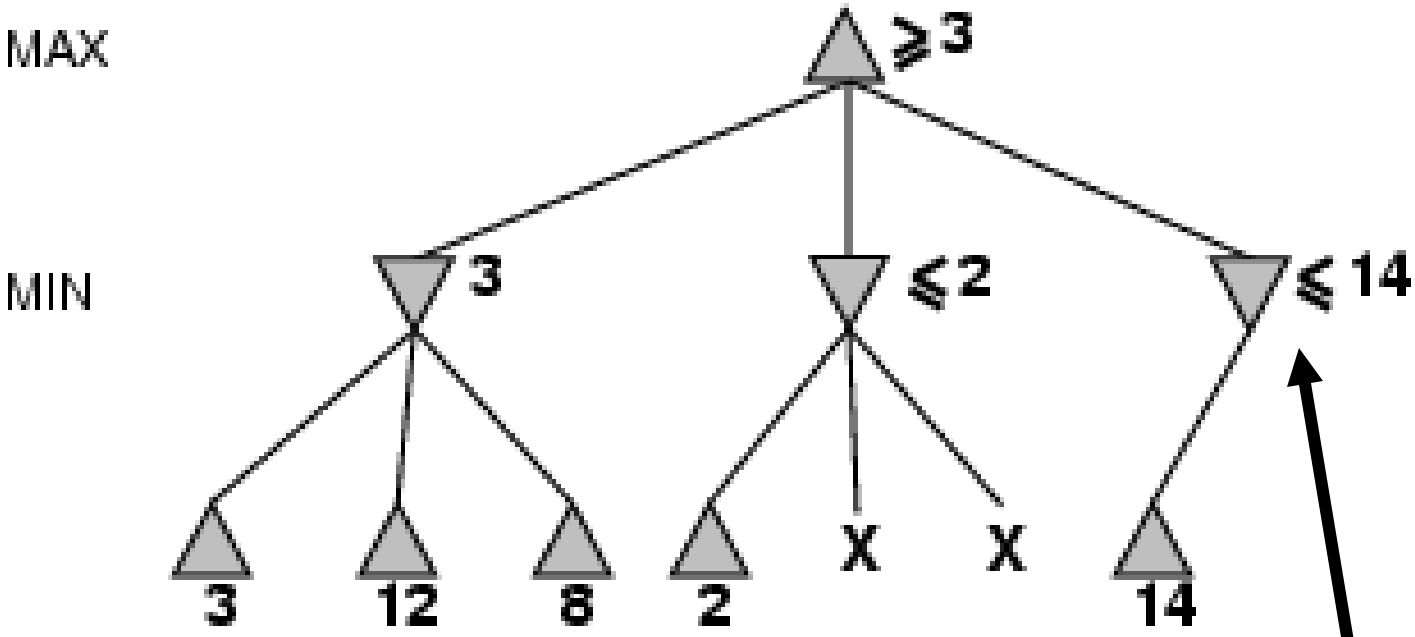
Ilustracija rada: α-β pruning



MAX zna da nova grana nikad neće biti bolja od 2 za njega.
Zato je *ignoriše*.

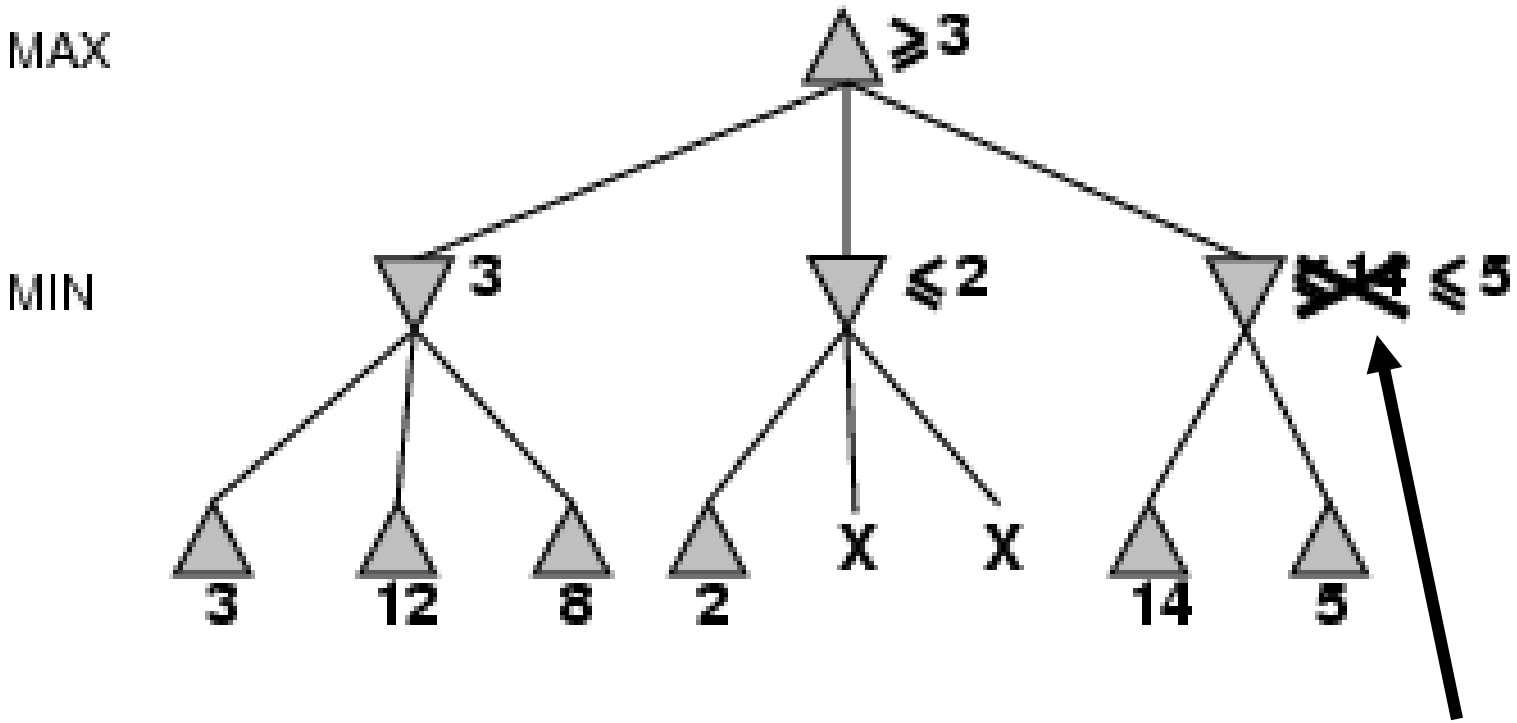
MIN može sigurno da odigra 2, a možda i bolje od toga (= manje)

Ilustracija rada: α-β pruning



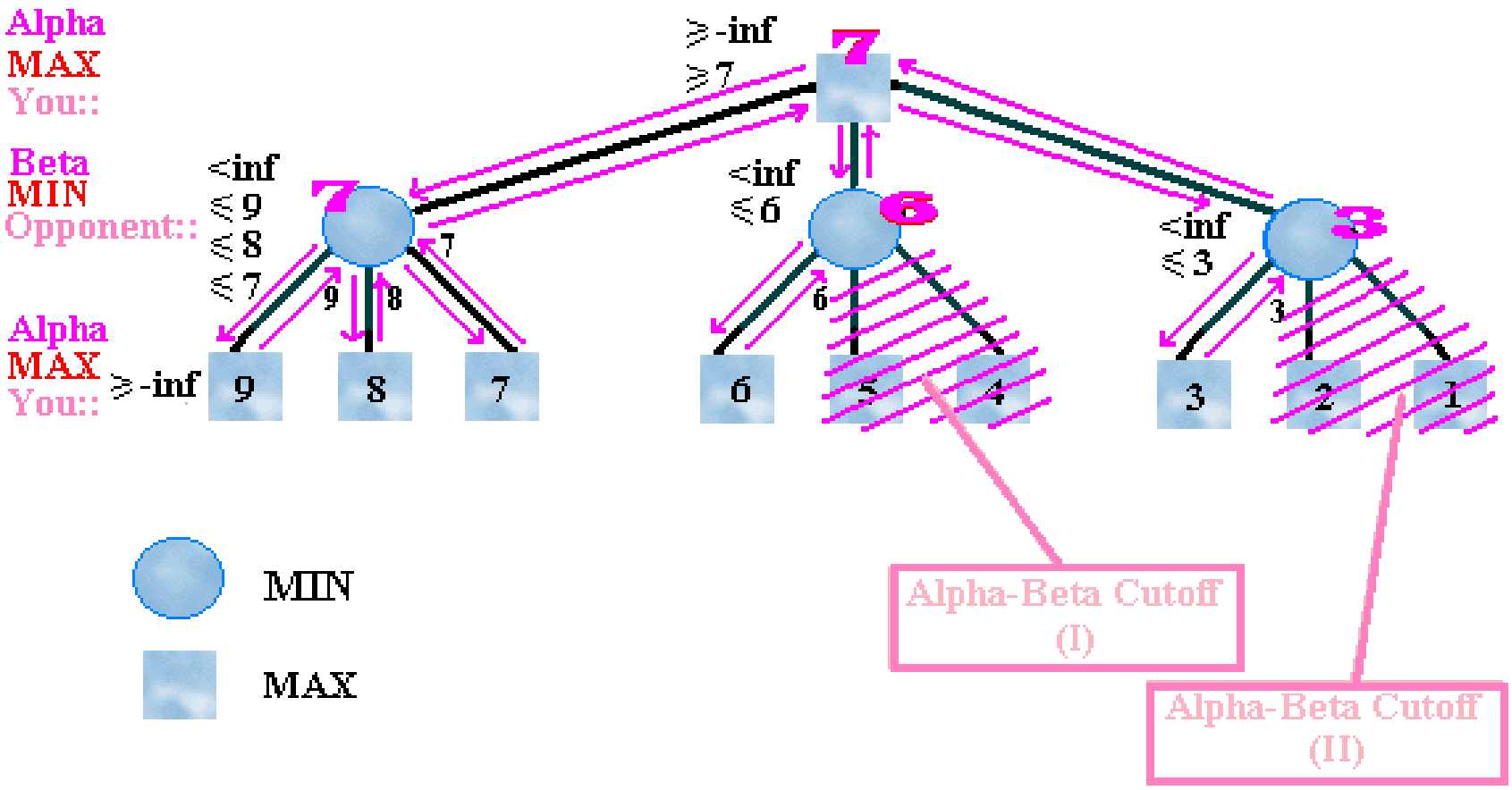
MIN ima potez koji ima vrednost 14 u ovoj grani (što je jako dobro za MAX!) pa MAX hoće da istraži ovu granu.

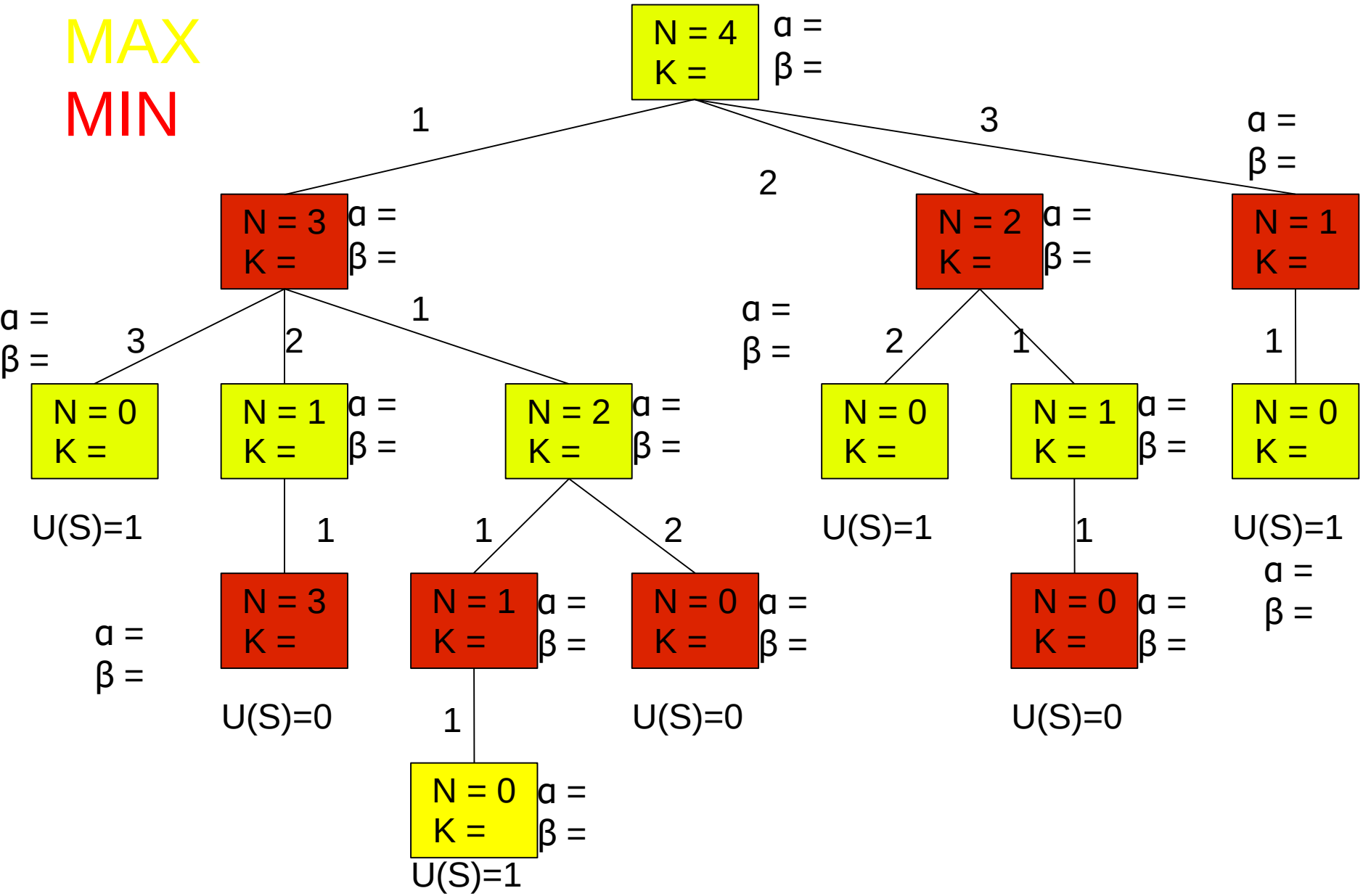
Ilustracija rada: α-β pruning



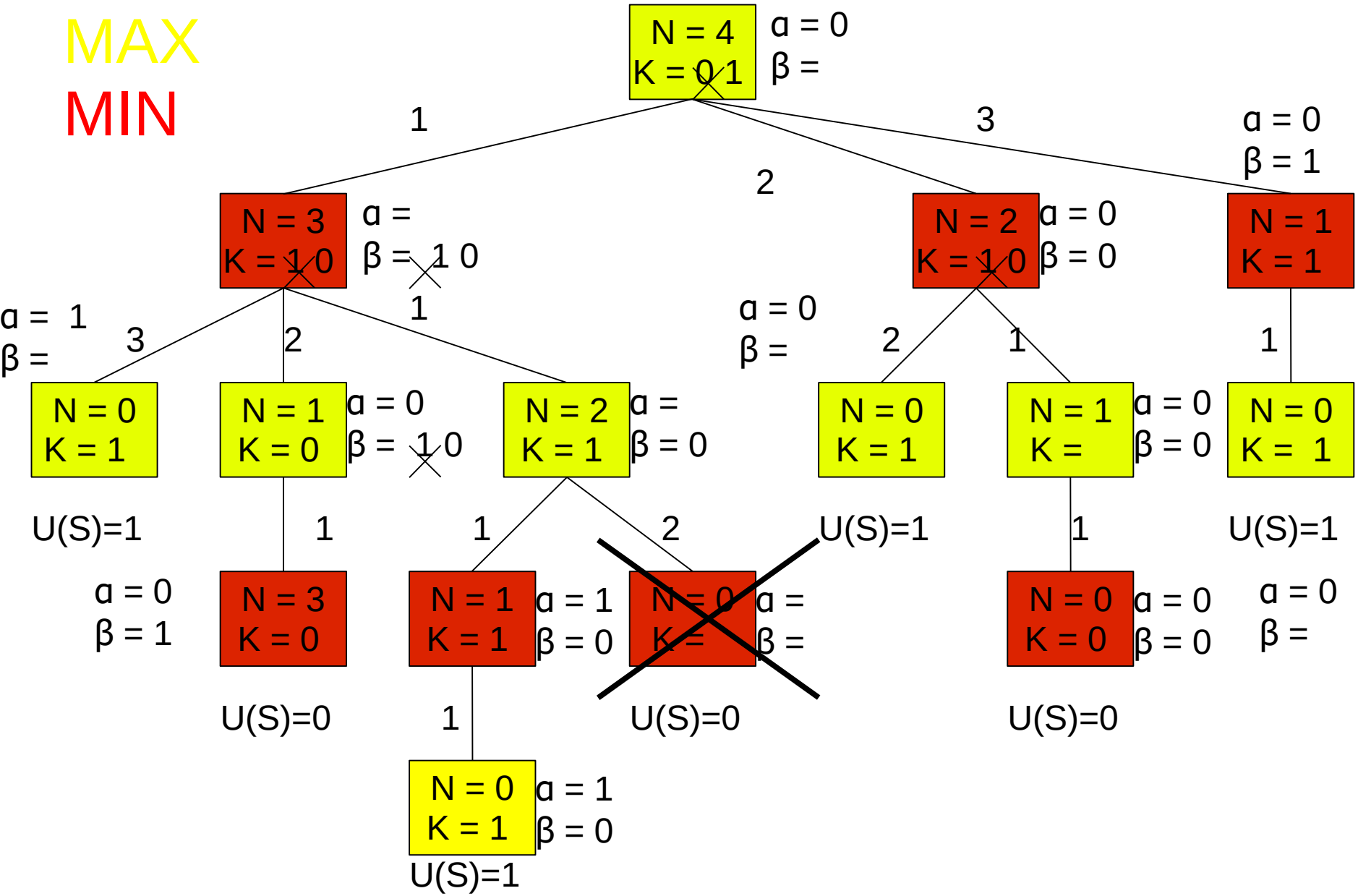
MIN sada ima potez koji vredi 5 u ovoj grani (što je i dalje dobro za MAX) pa MAX želi dalje da istraži ovu granu.

Primer: AlfaBeta odsecanje





Primer: AlfaBeta odsecanje



Primer: AlfaBeta odsecanje

Osobine α-β

- Odsecanje ne utiče na konačni rezultat.
- Dobar redosled poteza poboljšava efikasnost odsecanja

  (pogledati prethodni primer – poslednje odsecanje)
- Sa "perfektnim redosledom"
  vremenska kompleksnost = $O(b^{m/2})$

  → duplira dubinu traženja

PITANJA?

Dileme?

Komentari?