INTELIGENTNI SISTEMI

as. ms Vladimir Jocović
as. ms Adrian Milaković

TEORIJA IGARA

Sekvencijalne igre

03

„When you see a good move, look for a better one.“
- Emanuel Lasker

KARAKTERISTIKE IGARA

U nastavku će od interesa biti igre sa sledećim karakteristikama:

Sekvencijalne – odnosi se na tajming saopštavanja odluka igrača igre. Igrač saopštava donetu odluku nakon drugog igrača i na osnovu prethodno saopštene odluke drugog igrača. Sekvencijalne igre imaju vremensku osu i predstavljaju se stablima.

Savršene informacije – igrač ne zna koju je odluku doneo protivnički igrač dok je on sam ne saopšti, ali u svakom trenutku zna sve događaje koji su prethodili i doveli do trenutnog stanja u igri, uključujući i inicijalizacioni događaj (startno stanje igre).

Višepotezne – igrači donose više odluka u igri, u svakom potezu po jednu.

Zero-sum – osnovna karakteristika zero-sum igara je da je suma dobitaka svih igrača jednaka je 0.

U igrama će uvek učestovati najmanje **dva** igrača.

STABLO IGRE

Šta su osnovni elementi stabla igre?

Čvor – predstavlja jednu poziciju višepotezne igre.

Interni čvor - označava poziciju u igri u kojoj igrač na potezu i dalje može doneti odluku. Kvalitet ovakve pozicije po igrača se ne može precizno utvrditi.

Terminalni čvor (list) - označava poziciju kraja igre u kojoj igrač na potezu ne može doneti odluku. U ovakvim pozicijama se precizno može odrediti rezultat igre.

Grana – predstavlja jednu moguću odluku igrača koji je na potezu.

Nivo stabla – igrači igraju naizmenično (jedan za drugim) te svakom igraču odgovara jedan nivo stabla igre.

STABLO IGRE

Šta su osnovne odlike stabla igre?

Osnovne odlike stabla igre ukazuju na kompleksnost same igre.

Faktor grananja internih čvorova – označava broj mogućih odluka koje igrač na potezu može doneti. Faktor grananja je odličan indikator težine igre (veći faktor => teža igra).

Dubina stabla – za svaki čvor označava broj poteza koji dovodi do pozicije predstavljene tim čvorom u igri počevši od korena stabla (dubina nula). Veća dubina stabla ukazuje na kompleksiju igru.

Transpozicije – u nekim igrama je do iste pozicije (stanja) u igri moguće doći različitim sekvencama poteza igrača. Postojanje transpozicije u igri i korišćenje njenog postojanja u algoritmima pretrage značajno ubrzava sam algoritam.

MINIMAX IDEJA

Cilj svakog racionalnog igrača u igri je da donese odluku koja je najbolja po njega, tj. da od svih poteza koji su mu na raspolaganju izabere onaj koji mu donosi najveći dobitak.

Da bi igrač znao koju odluku da donese on mora da proceni kakav je efekat donošenja njegove odluke na drugog igrača. U najgorem slučaju po prvog igrača, drugi igrač je takođe racionalan i želeće da donese odluku koja je najbolja po njega.

Kako su nam od interesa Zero-sum igre, to znači da je **najbolja** odluka po drugog igrača u isto vreme i **najgora** odluka po prvog igrača, jer je dobitak drugog igrača ekvivalentan gubitku prvog. Stoga će prvi igrač, kao racionalan, želeti da **min**imizuje svoj **maks**imalni potencijalni gubitak. Otuda ime ovog algoritma.

MINIMAX IDEJA

U igri sa 2 igrača jednog ćemo označiti kao MAX, a drugog kao MIN.

Ukoliko su igraču MAX na raspolaganju odluke A, B, C i donose mu dobitak gain(A), gain(B) i gain(C), respektivno, onda je igraču MAX cilj da donese odluku koja mu donosi najveći dobitak:

max(gain(A), gain(B), gain(C))

Ukoliko su igraču MIN na raspolaganju odluke X, Y, Z i igraču MAX donose dobitak gain(X), gain(Y) i gain(Z), respektivno, onda je igraču MIN cilj da donese odluku koja je najbolja po njega, a najgora po MAX igrača:

min(gain(X), gain(Y), gain(Z))

MINIMAX ALGORITAM

```
def minimax(node, player):
    if is_terminal_node(node):
        return node_evaluation(node)

    if player == Player.MAX:
        score = -math.inf
        for succ in node.successors():
            score = max(score, minimax(succ, Player.MIN))
        return score
    else:
        score = +math.inf
        for succ in node.successors():
            score = min(score, minimax(succ, Player.MAX))
        return score
```

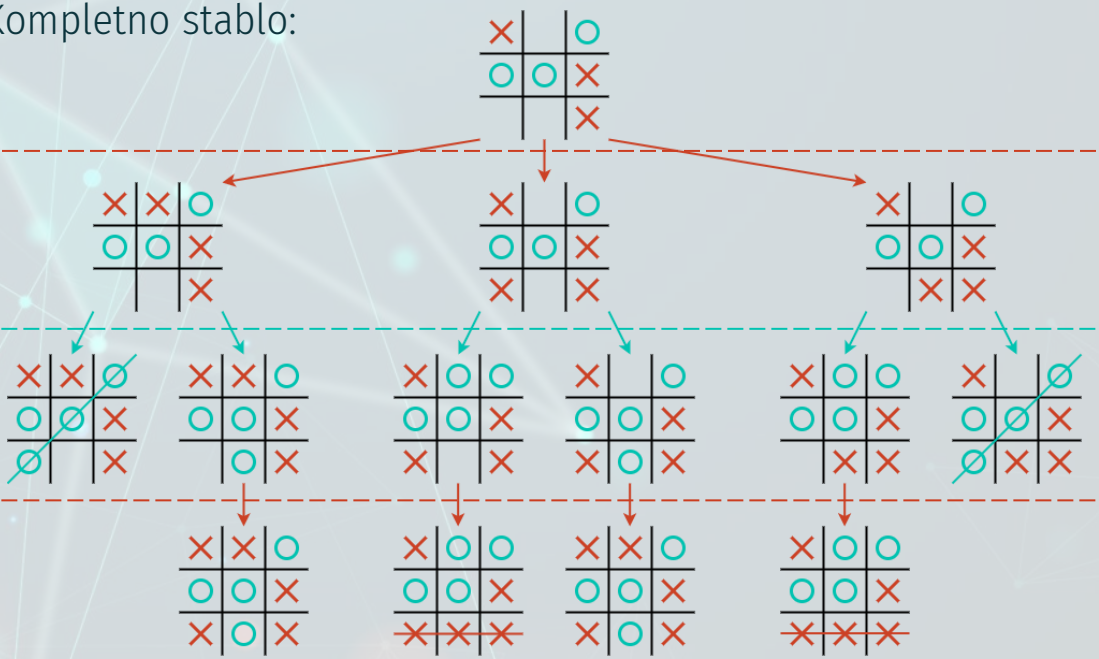
Zadatak 1 – XO (Minimax)

★☆☆☆☆

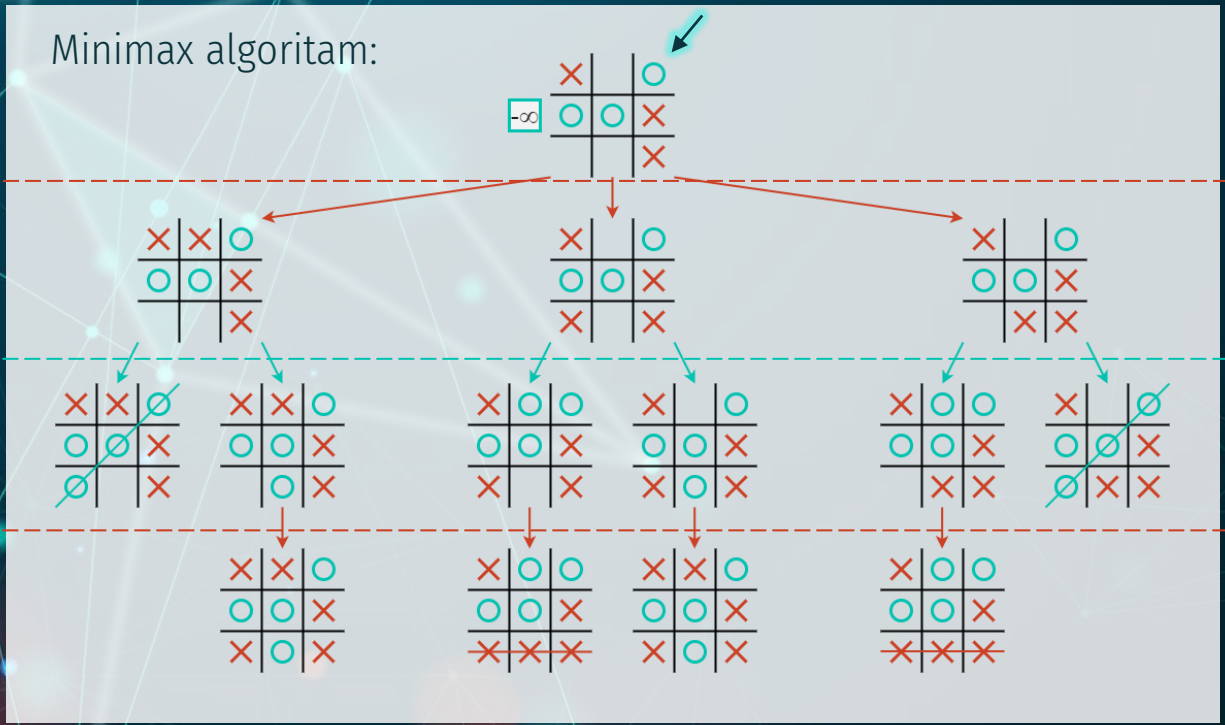
Data je jedna pozicija u jednoj partiji igre iks-oks. Potrebno je razviti kompletno stablo igre počevši od date pozicije kao startne. Primenom **minimax** algoritma odrediti funkciju procene za svaki čvor, konačan ishod igre i sekvencu poteza koja će biti odigrana do kraja ukoliko su oba igrača racionalna. Vrednost funkcije procene terminalne pozicije označiti sa +1 ukoliko je pobednik X, -1 ukoliko je pobednik O i 0 ukoliko je ishod igre nerešen. Na potezu je igrač X. Naslednike čvora razvijati dodajući znakove na slobodna polja po redovima.

Zadatak 1 – Rešenje

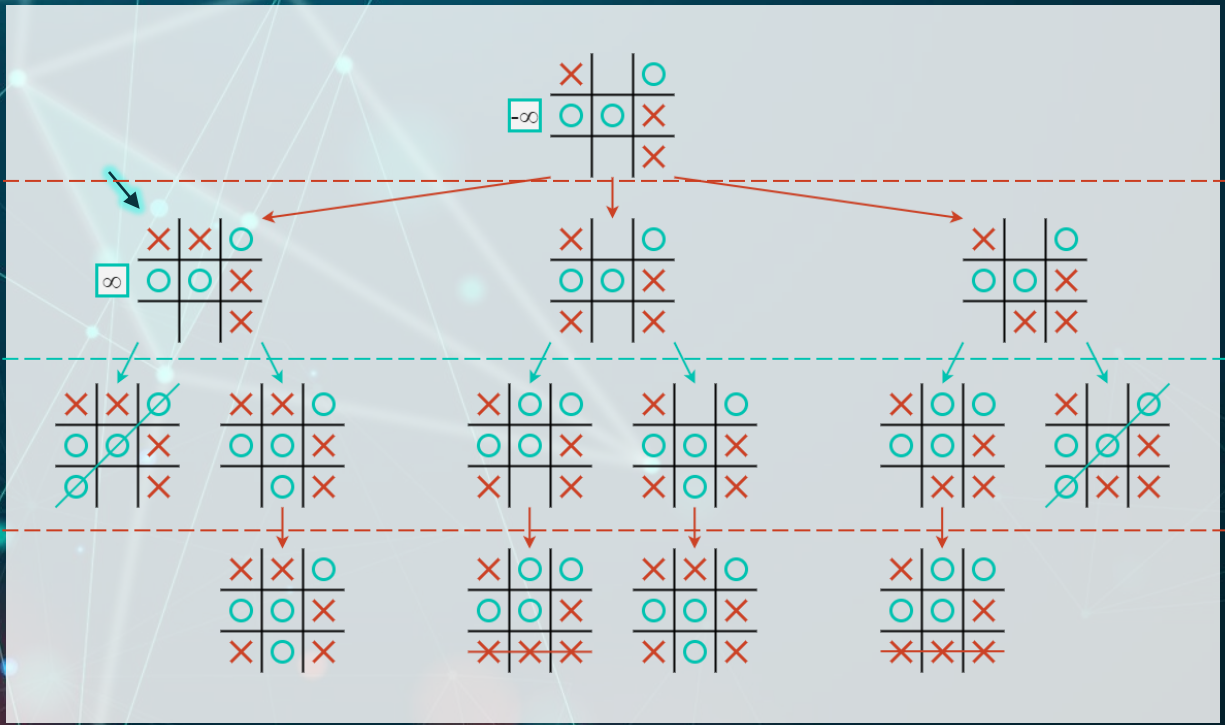
Kompletno stablo:



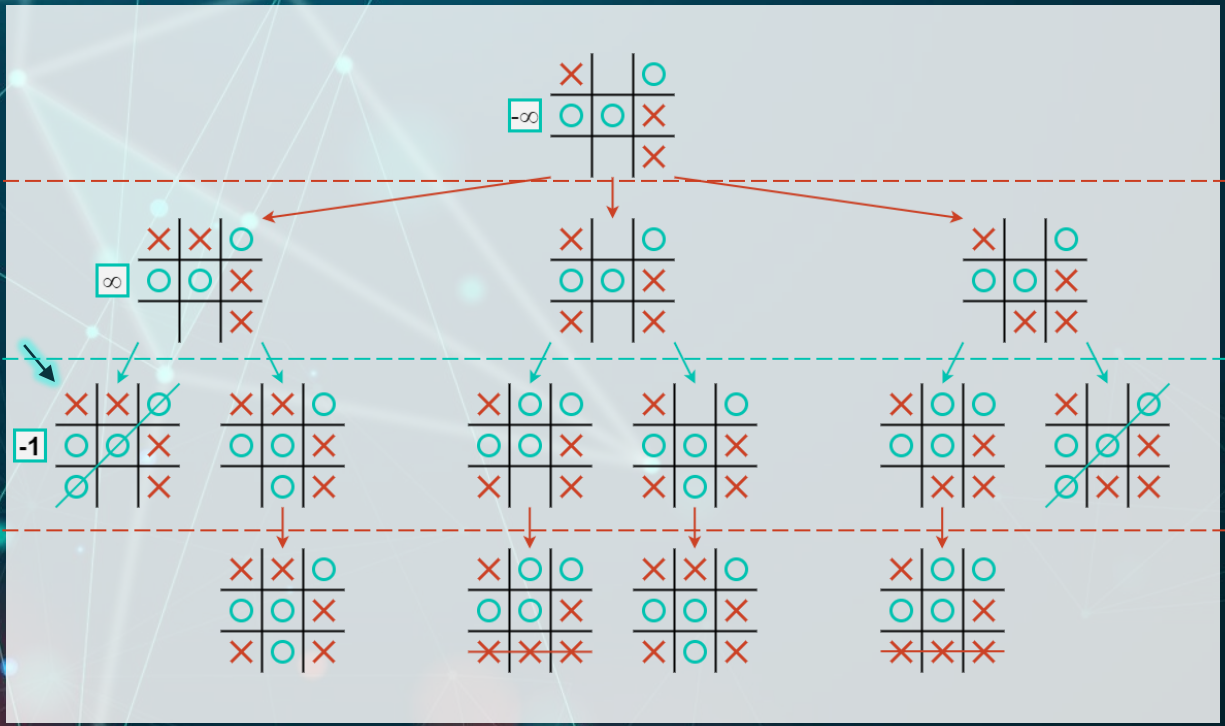
Zadatak 1 – Rešenje



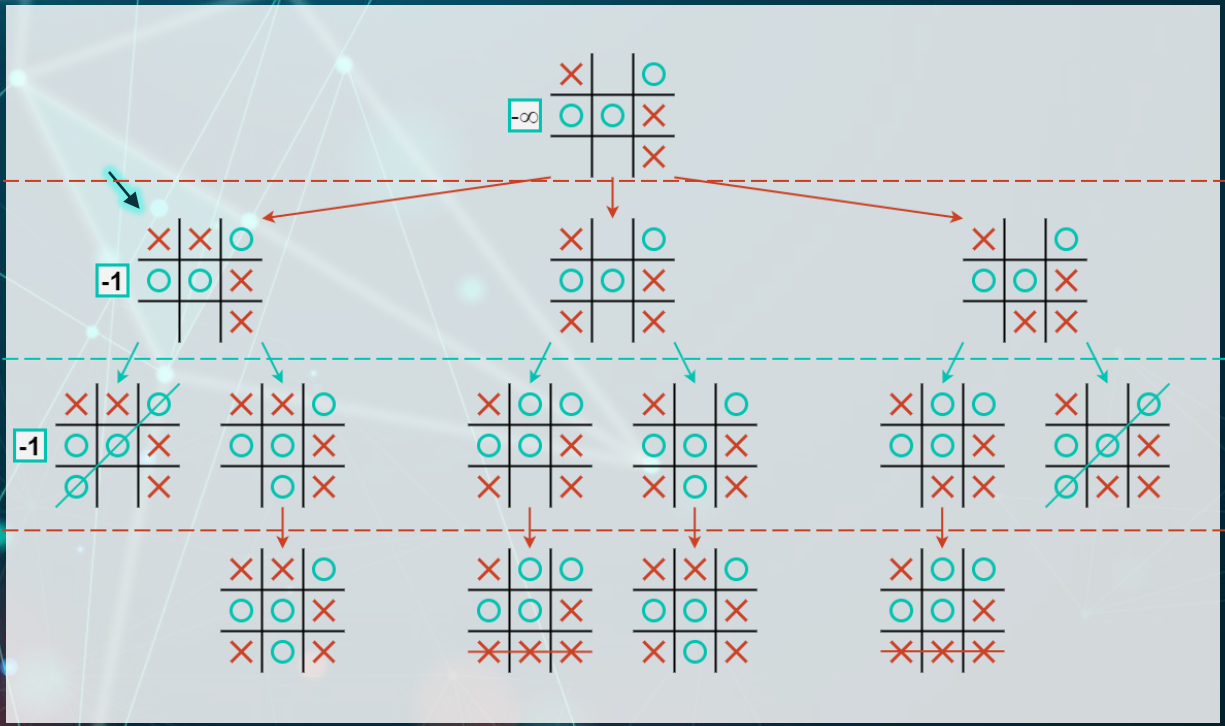
Zadatak 1 – Rešenje



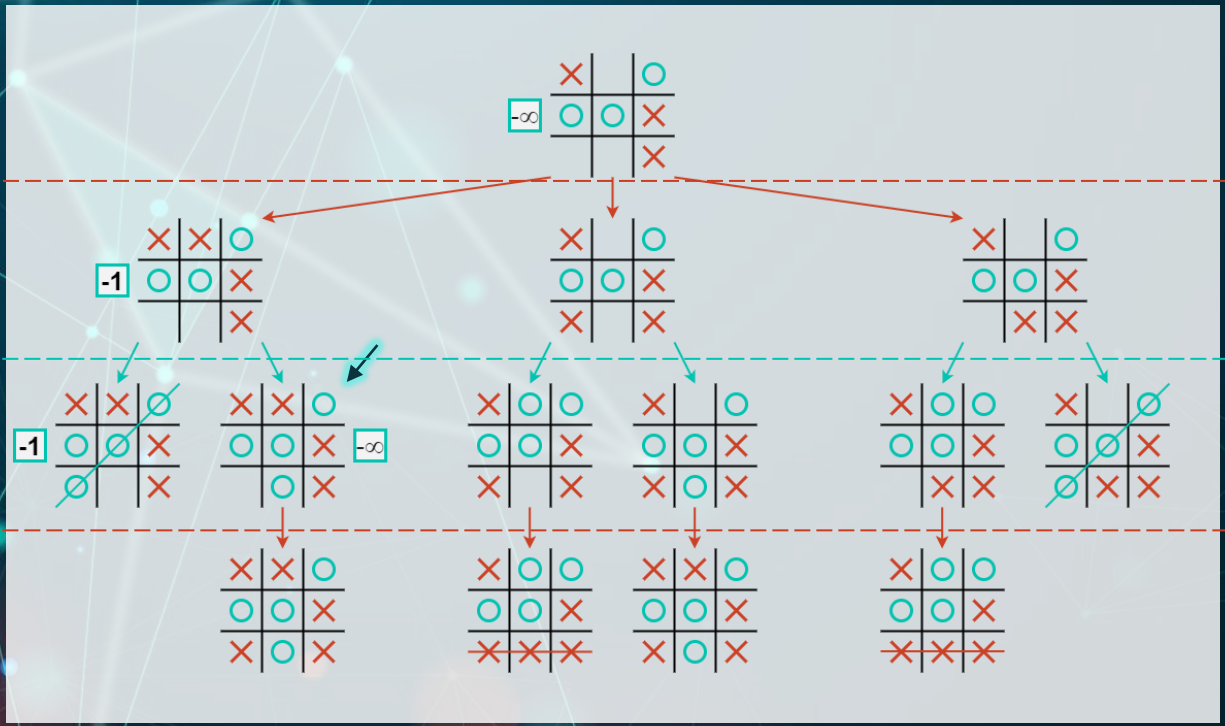
Zadatak 1 – Rešenje



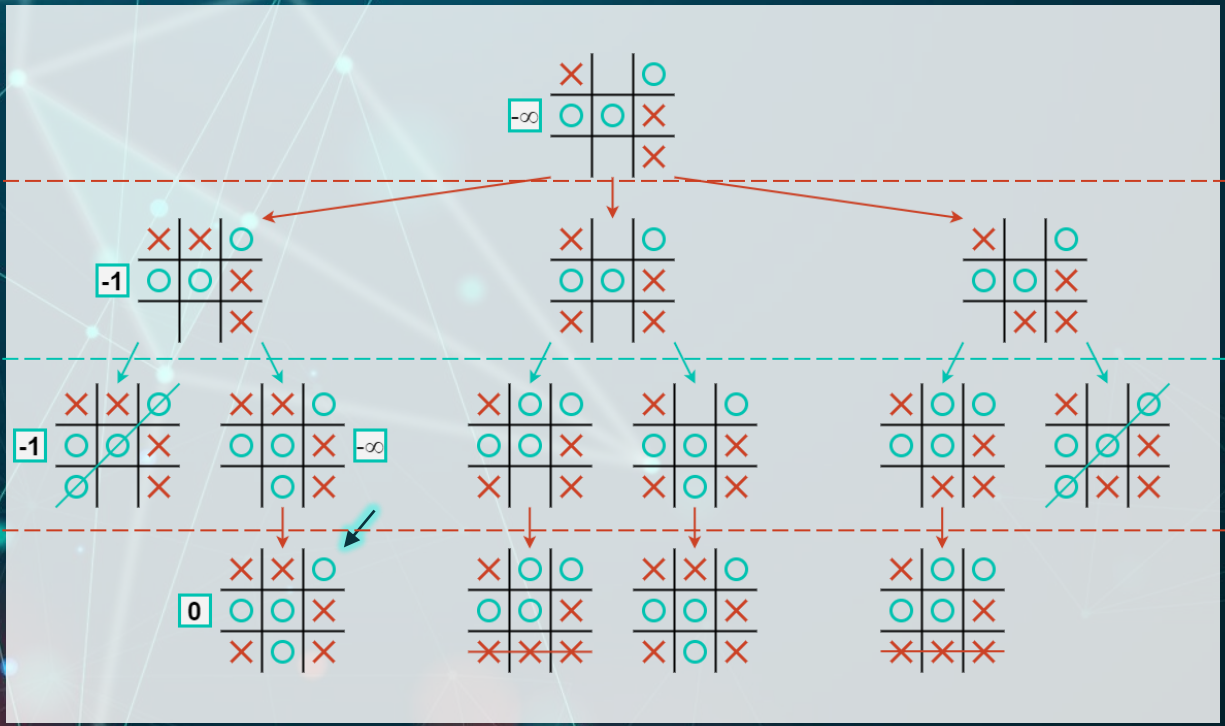
Zadatak 1 – Rešenje



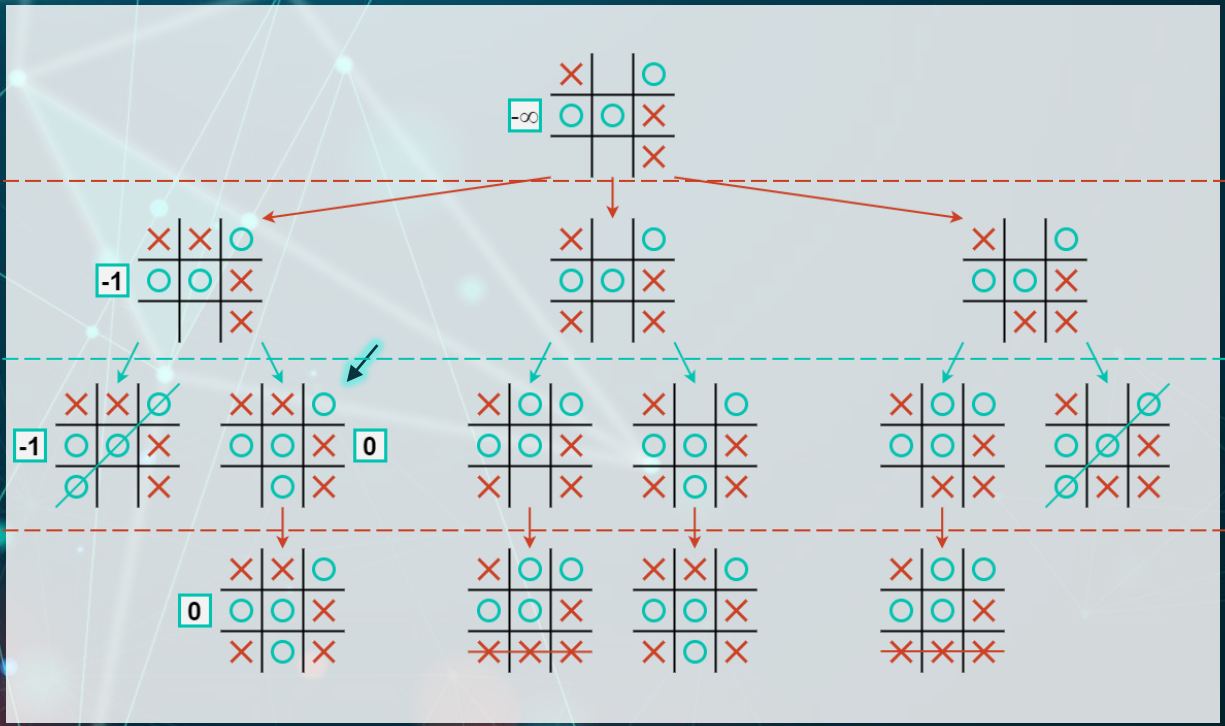
Zadatak 1 – Rešenje



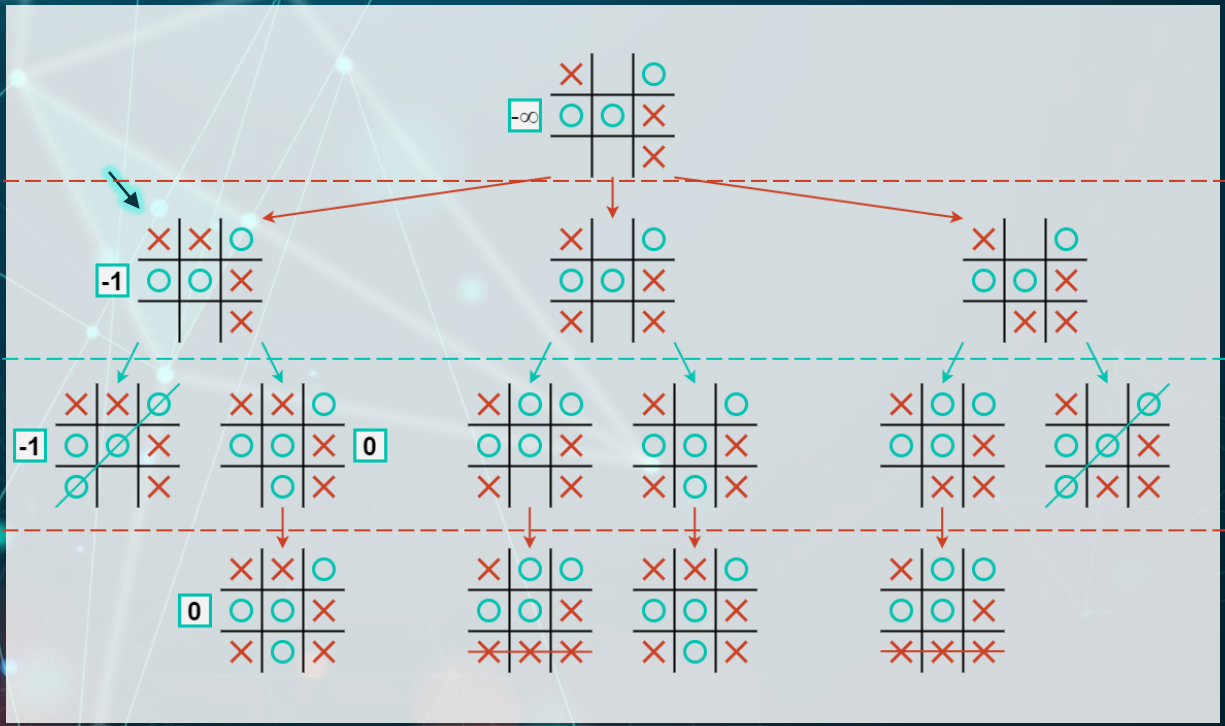
Zadatak 1 – Rešenje



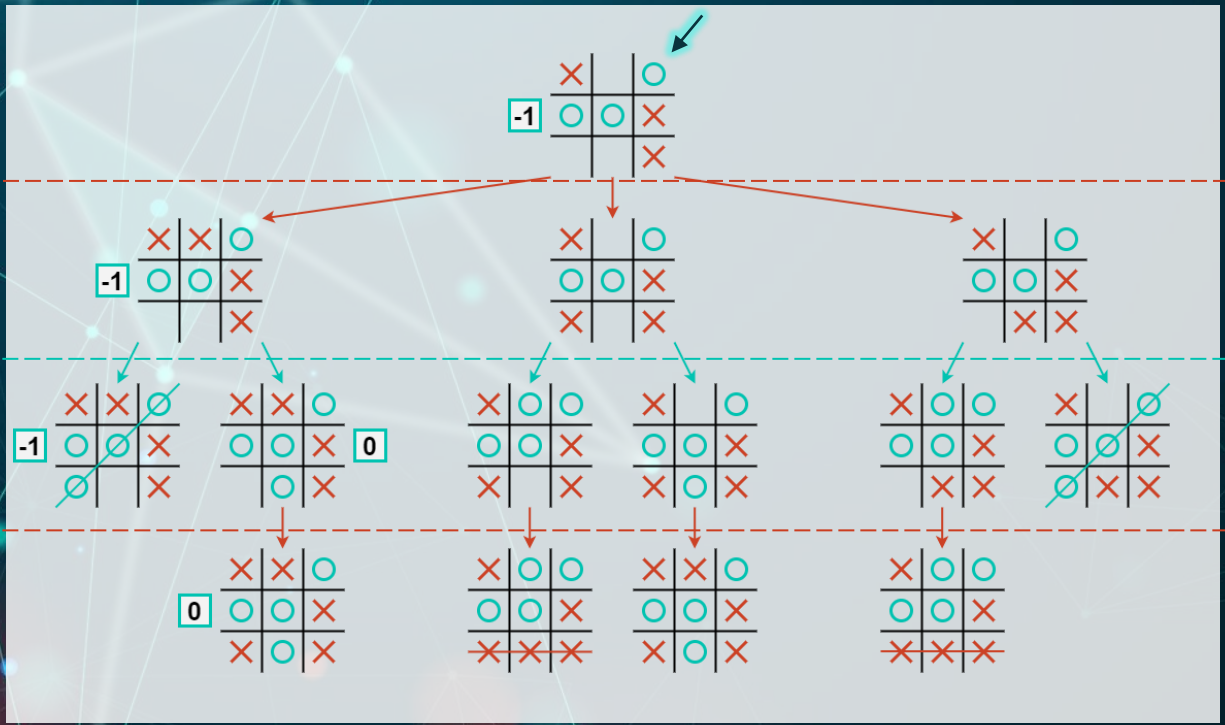
Zadatak 1 – Rešenje



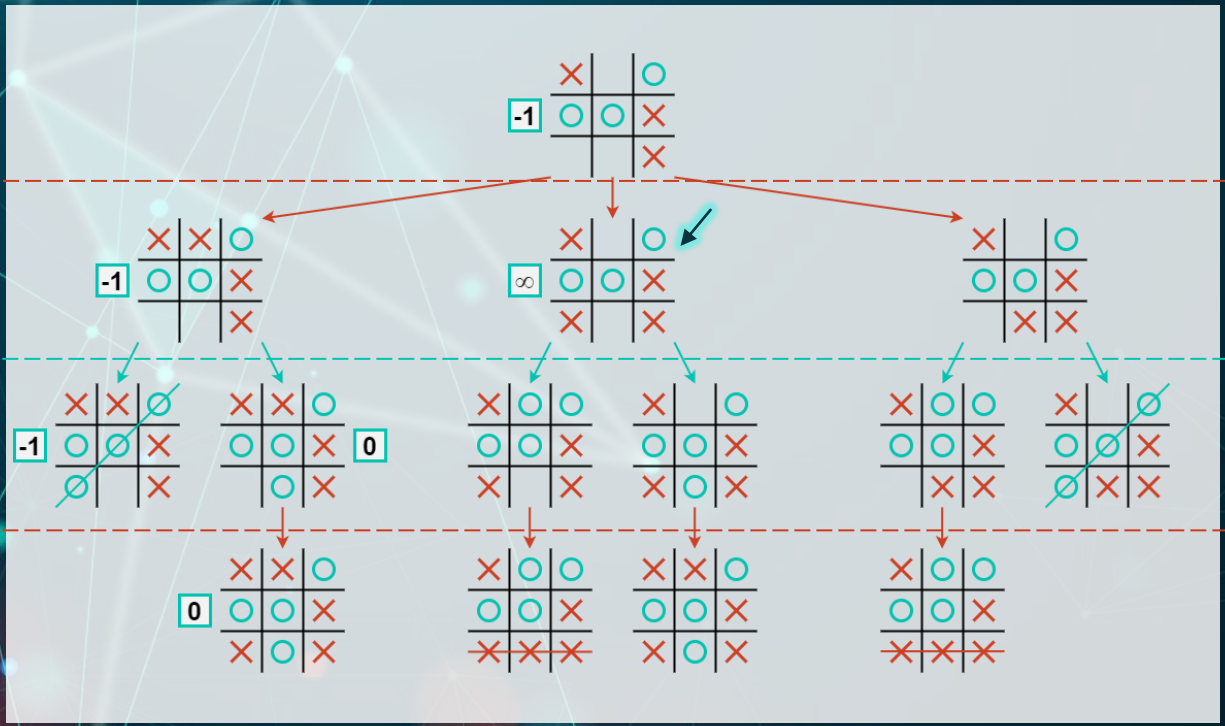
Zadatak 1 – Rešenje



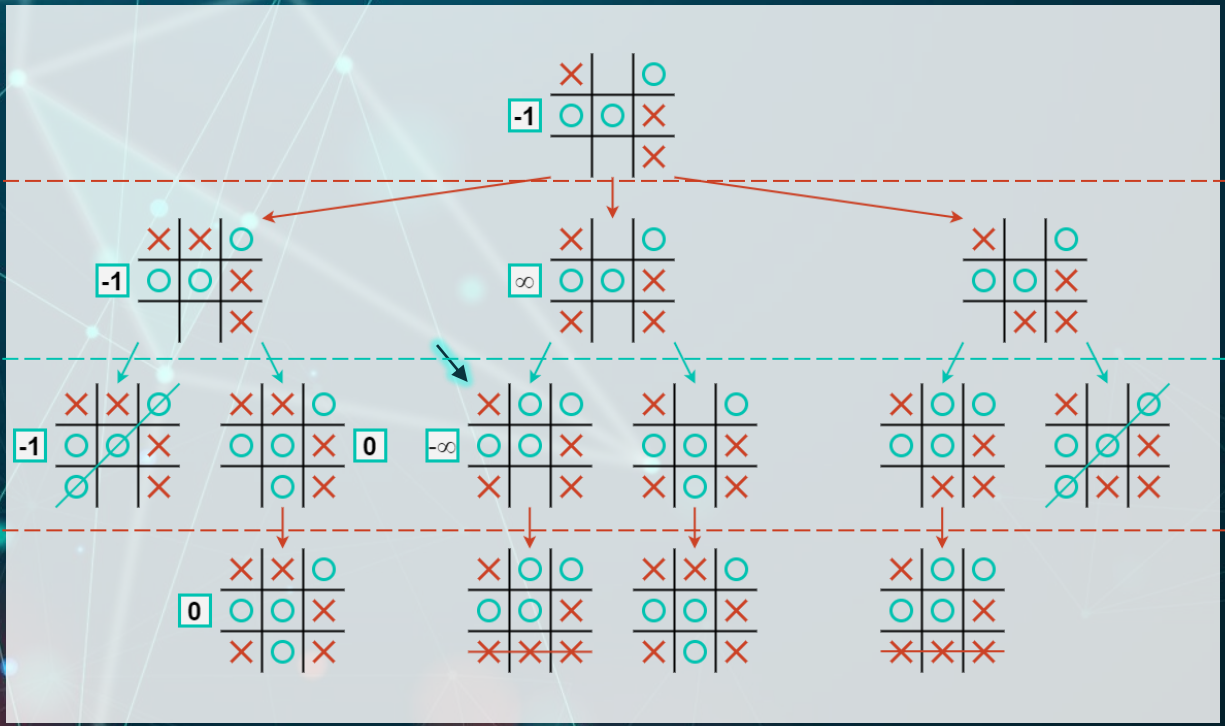
Zadatak 1 – Rešenje



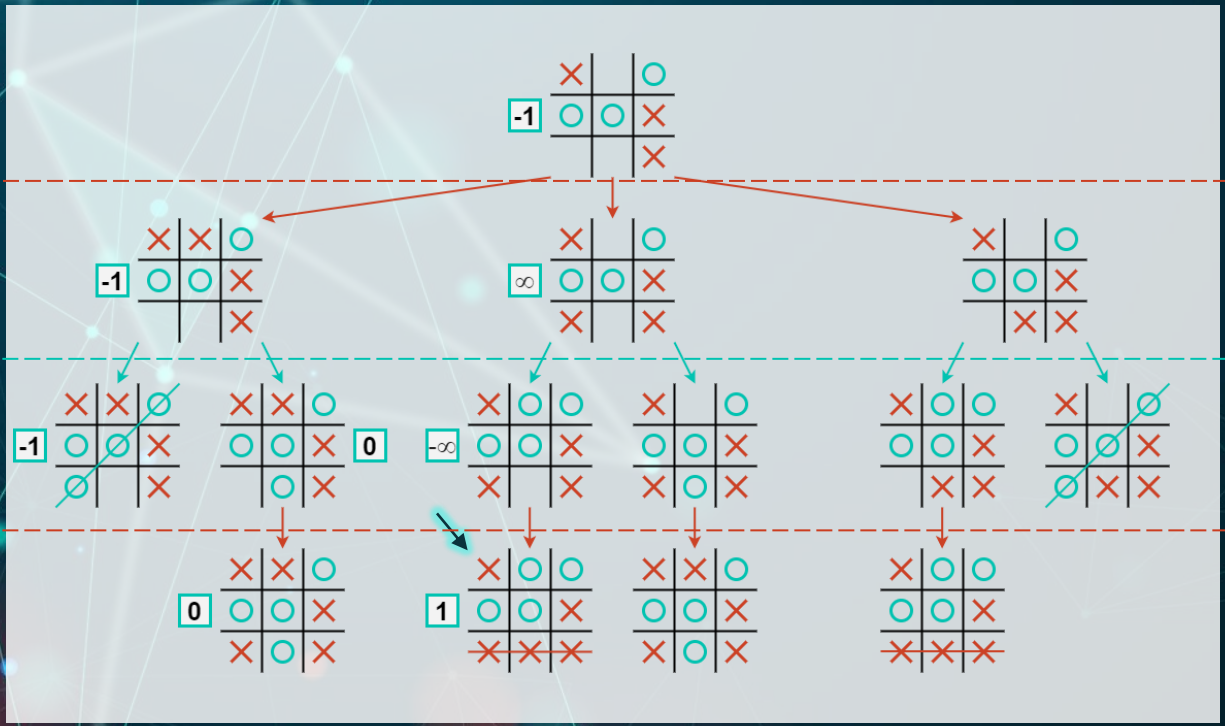
Zadatak 1 – Rešenje



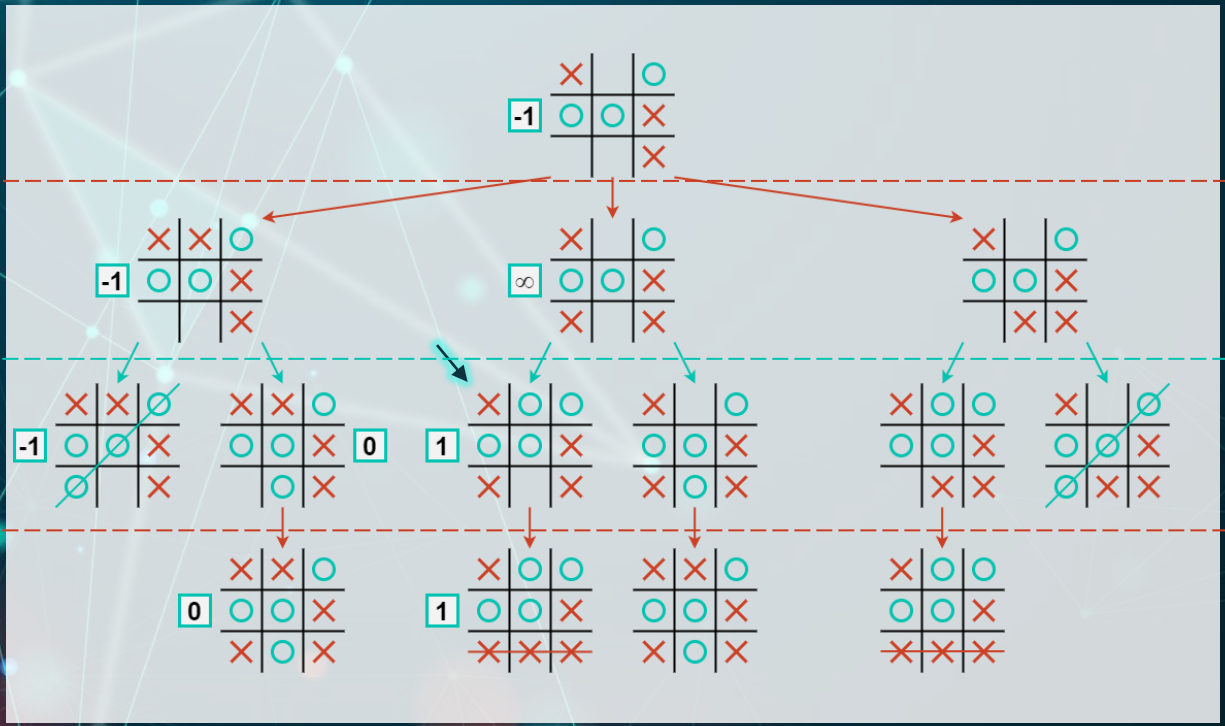
Zadatak 1 – Rešenje



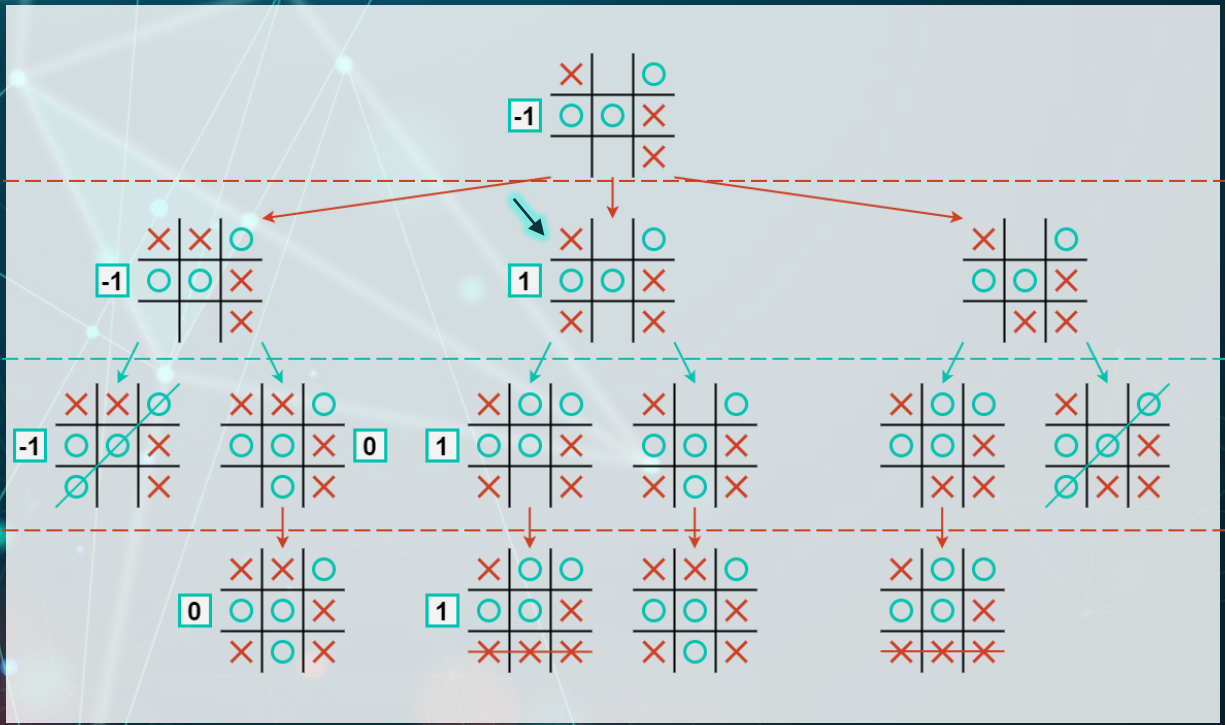
Zadatak 1 – Rešenje



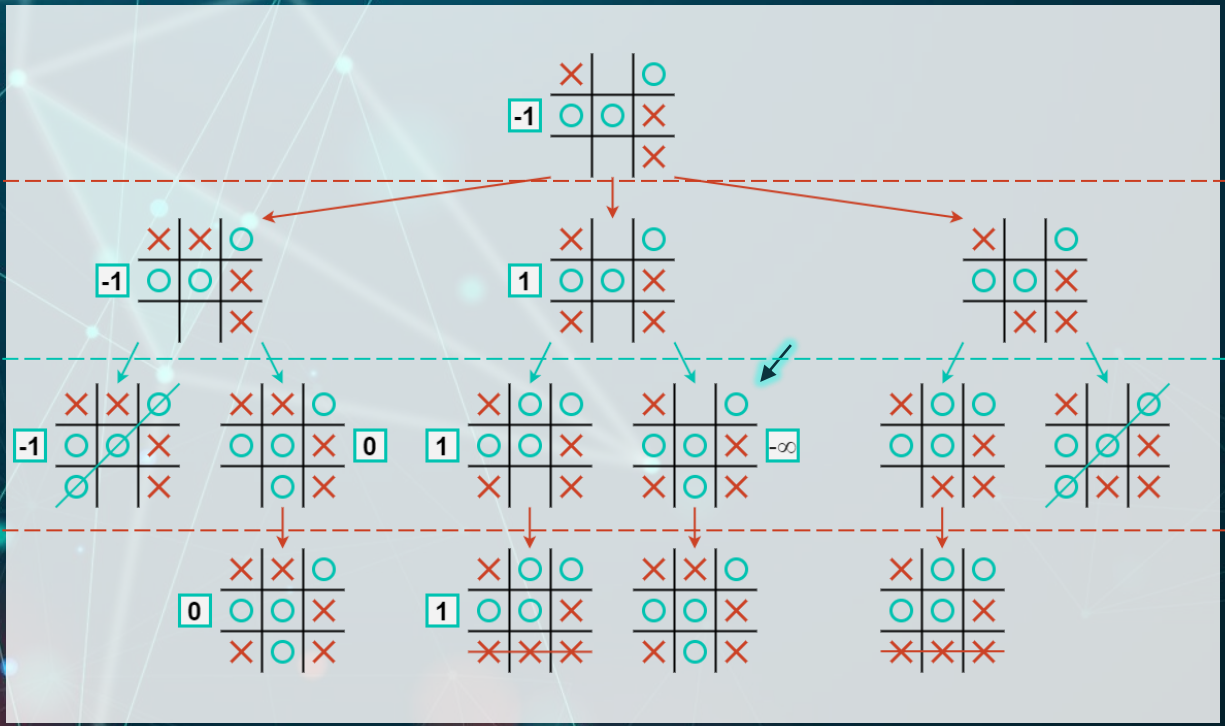
Zadatak 1 – Rešenje



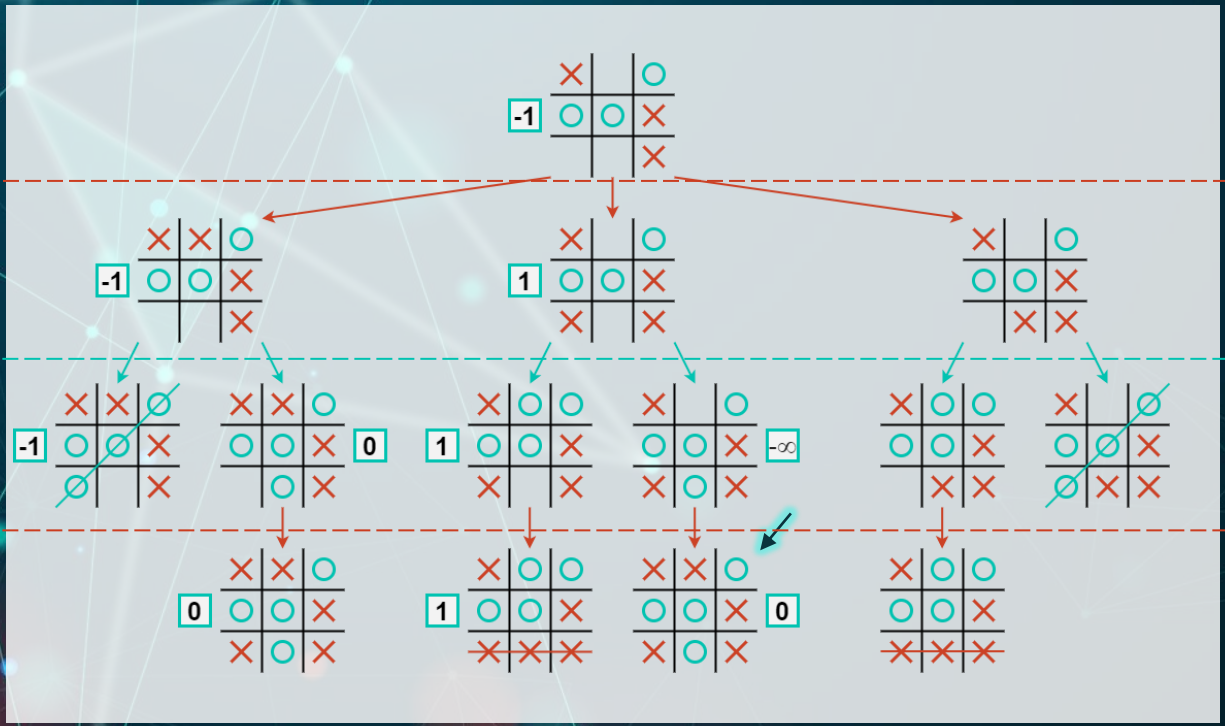
Zadatak 1 – Rešenje



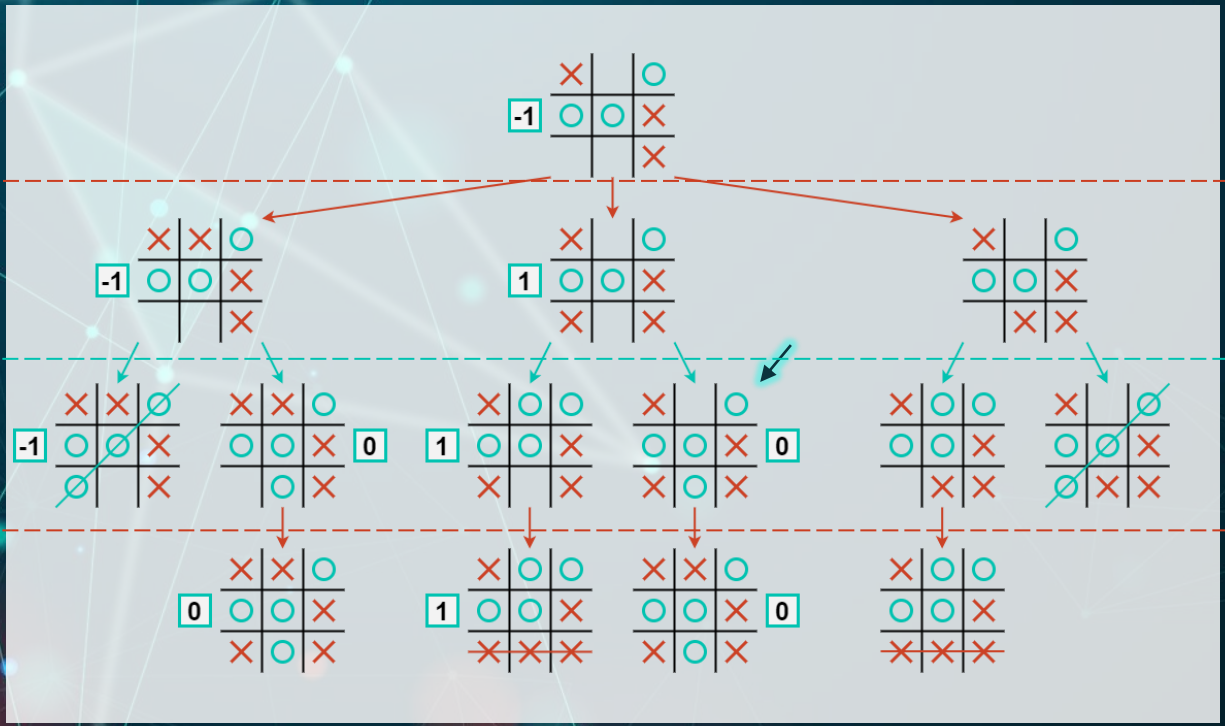
Zadatak 1 – Rešenje



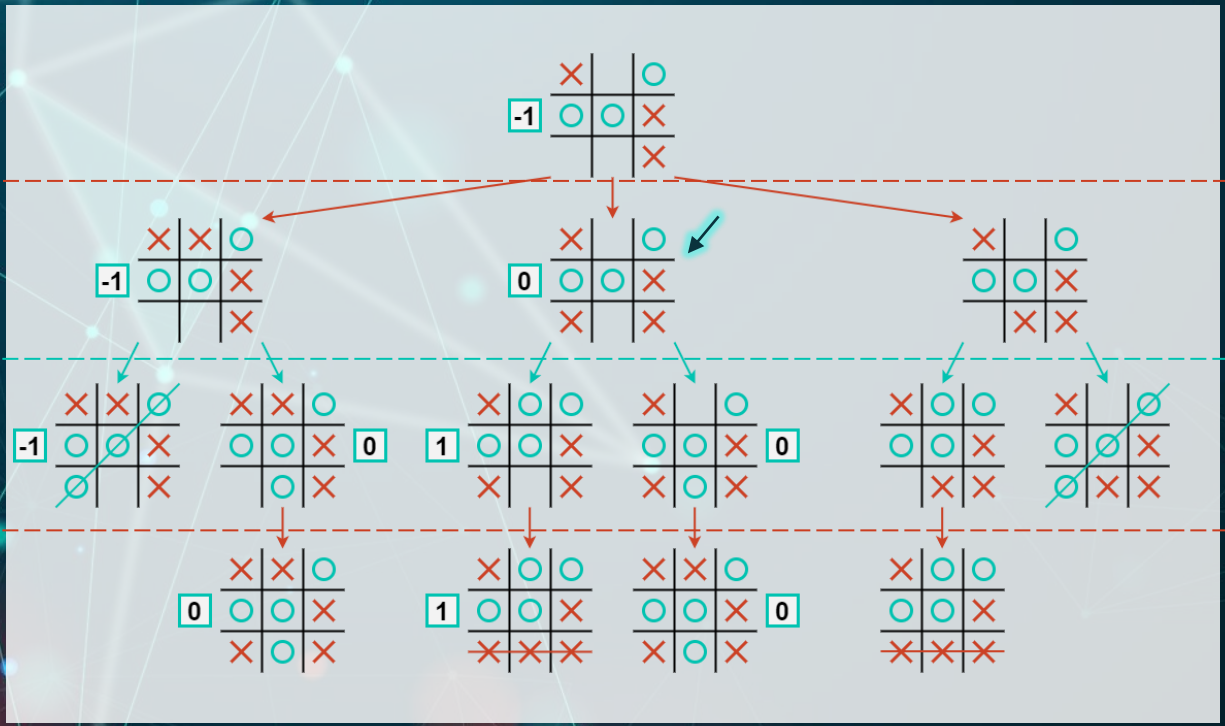
Zadatak 1 – Rešenje



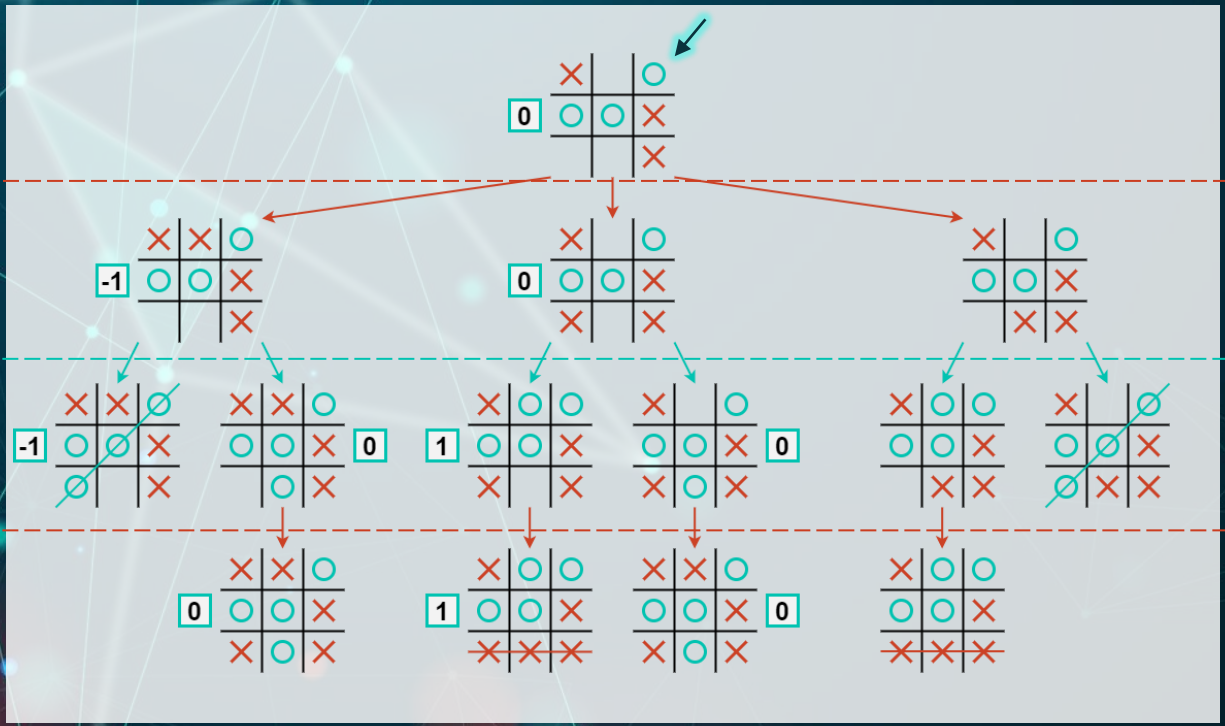
Zadatak 1 – Rešenje



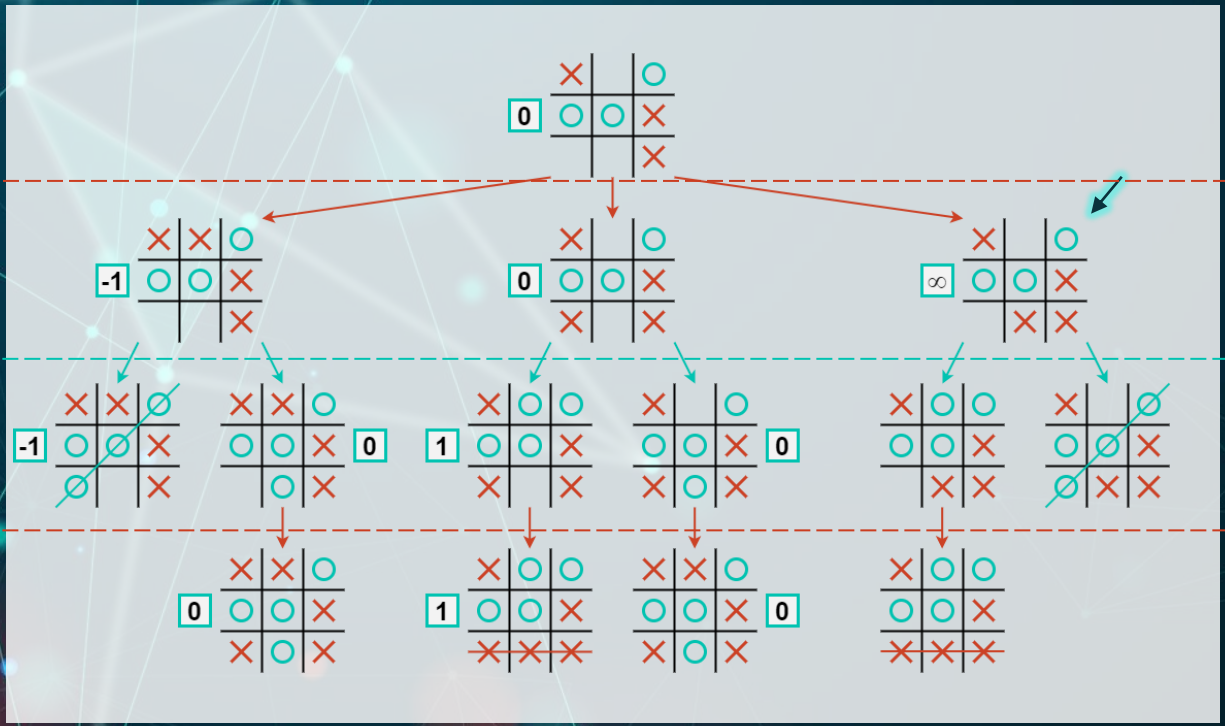
Zadatak 1 – Rešenje



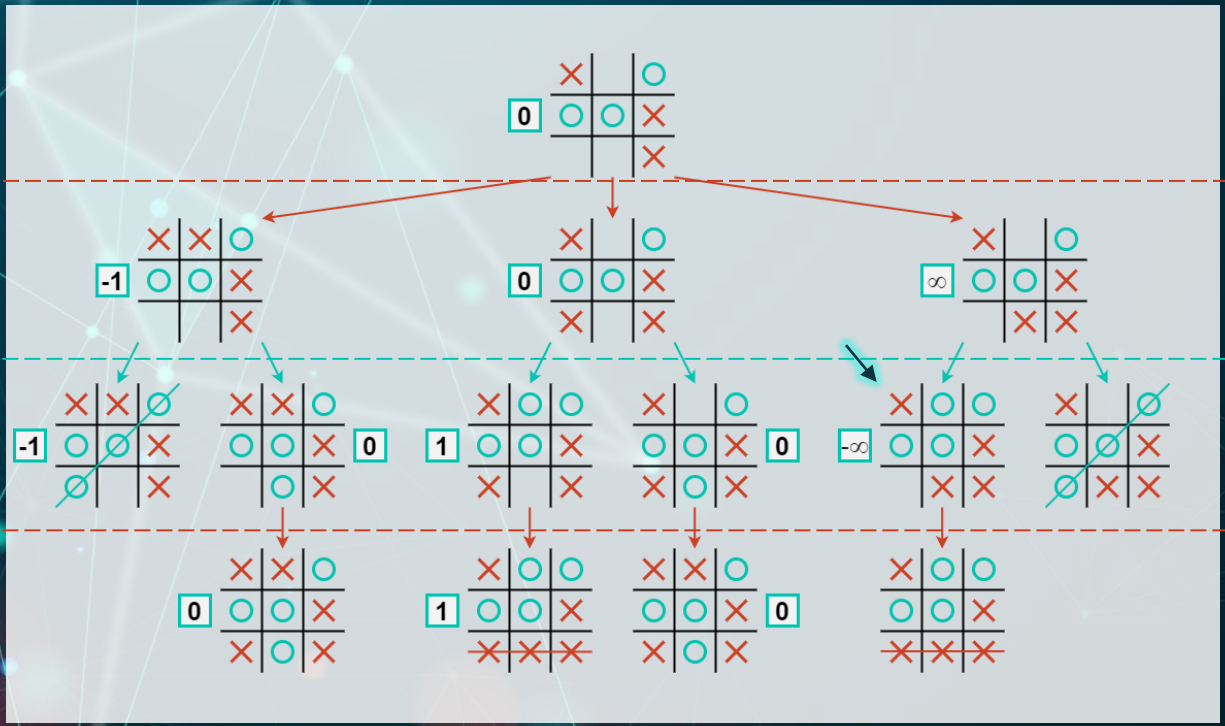
Zadatak 1 – Rešenje



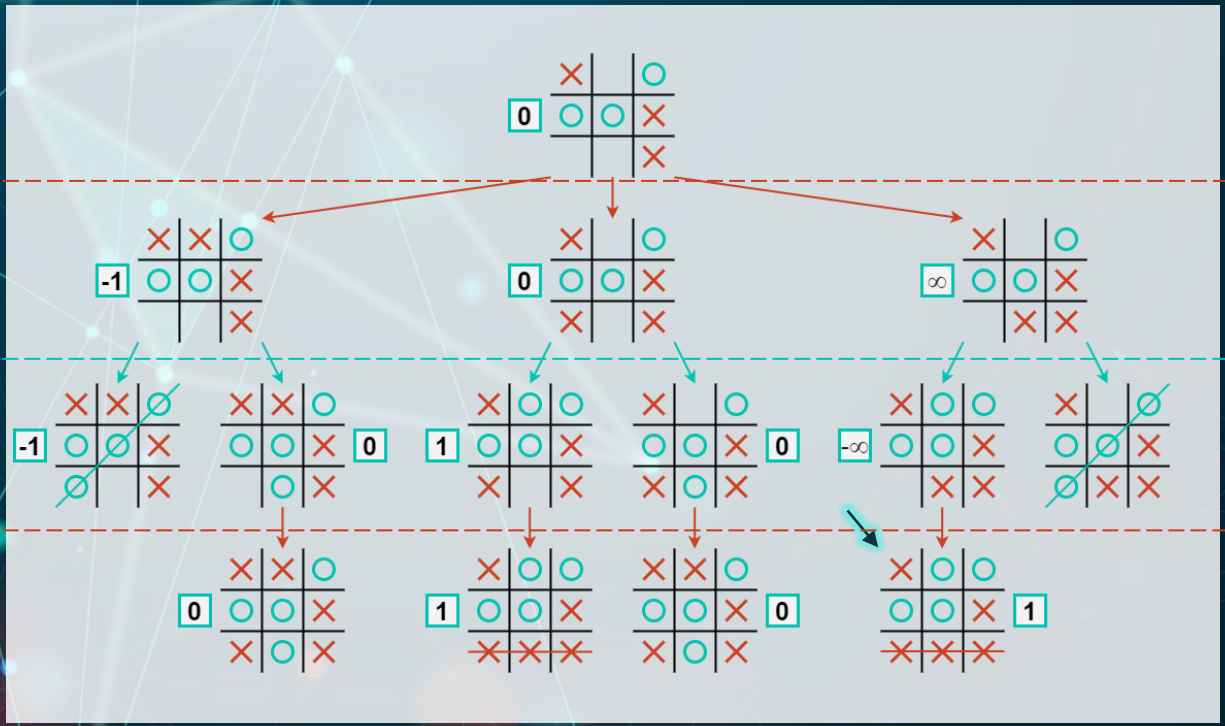
Zadatak 1 – Rešenje



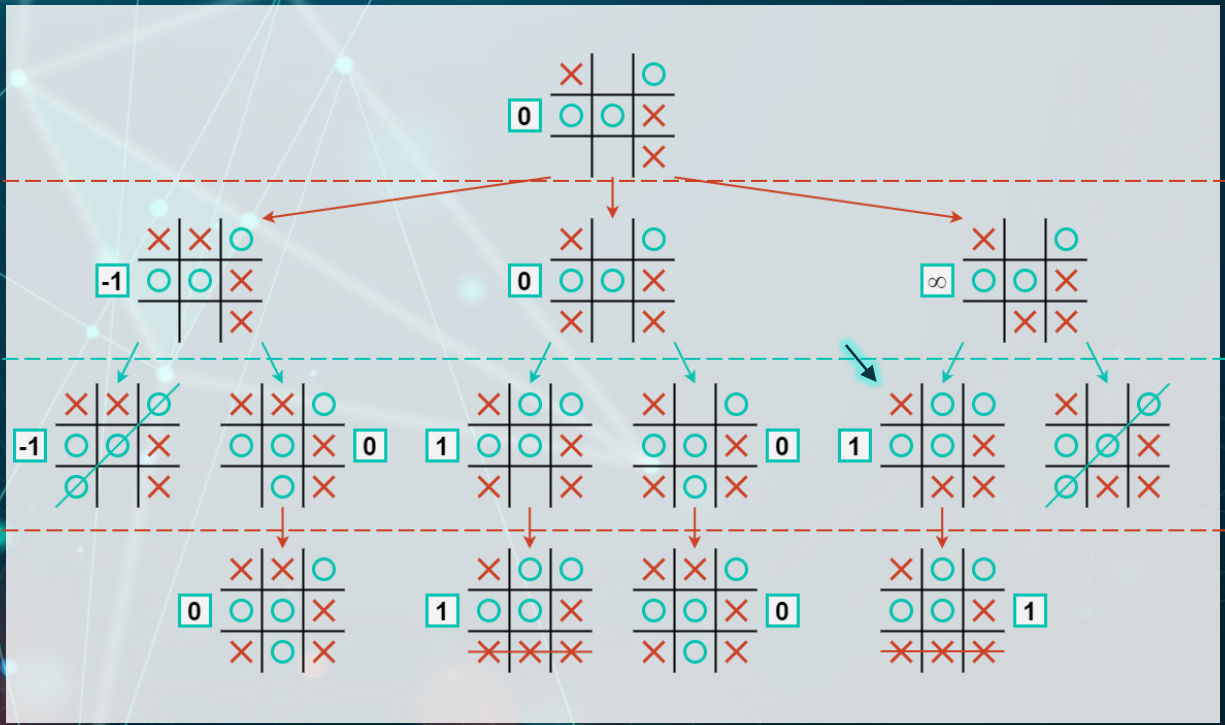
Zadatak 1 – Rešenje



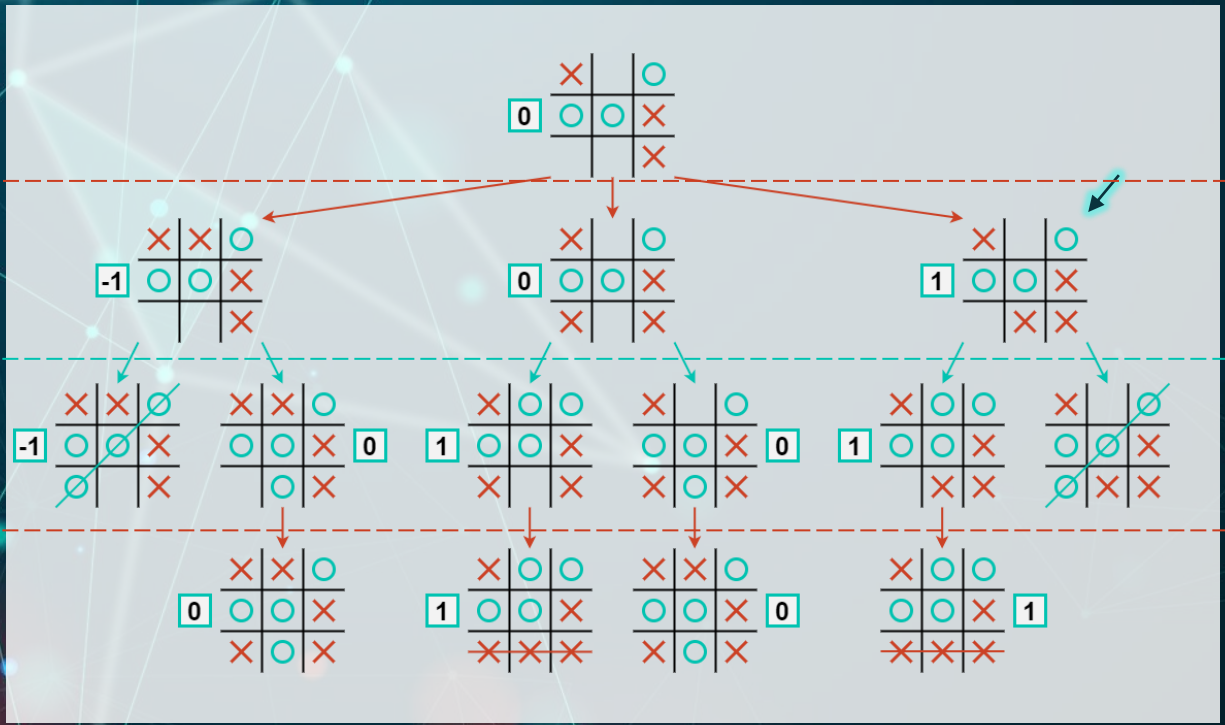
Zadatak 1 – Rešenje



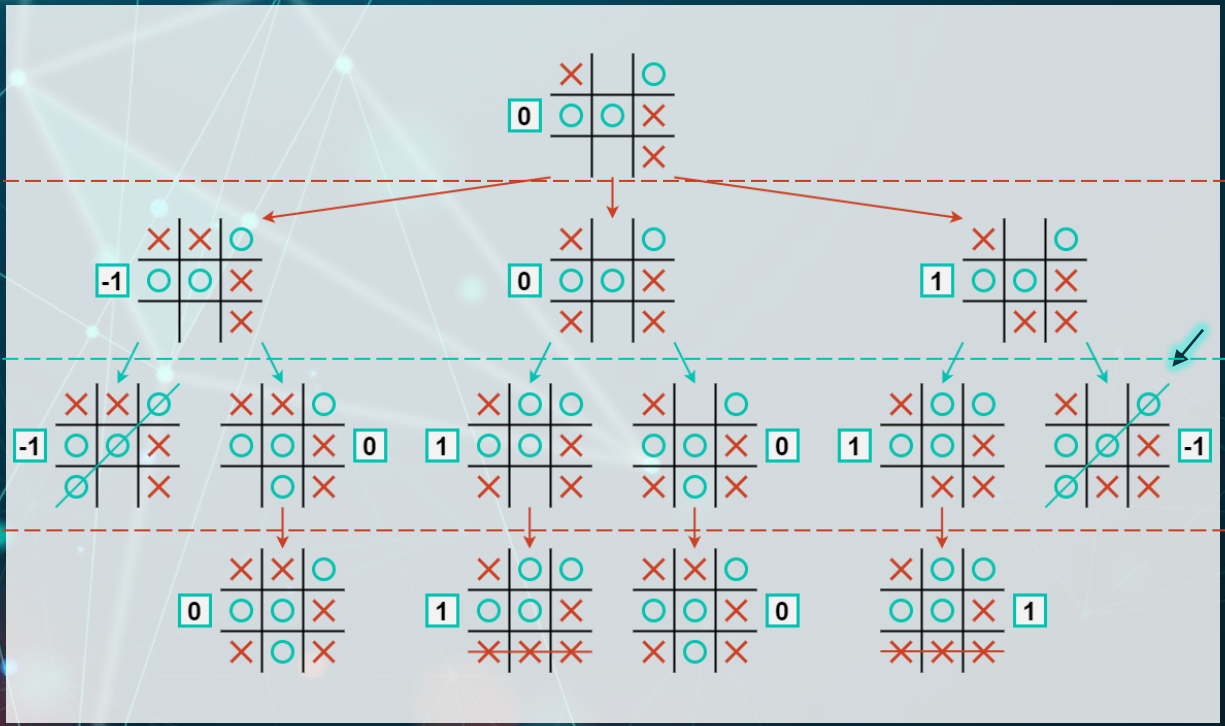
Zadatak 1 – Rešenje



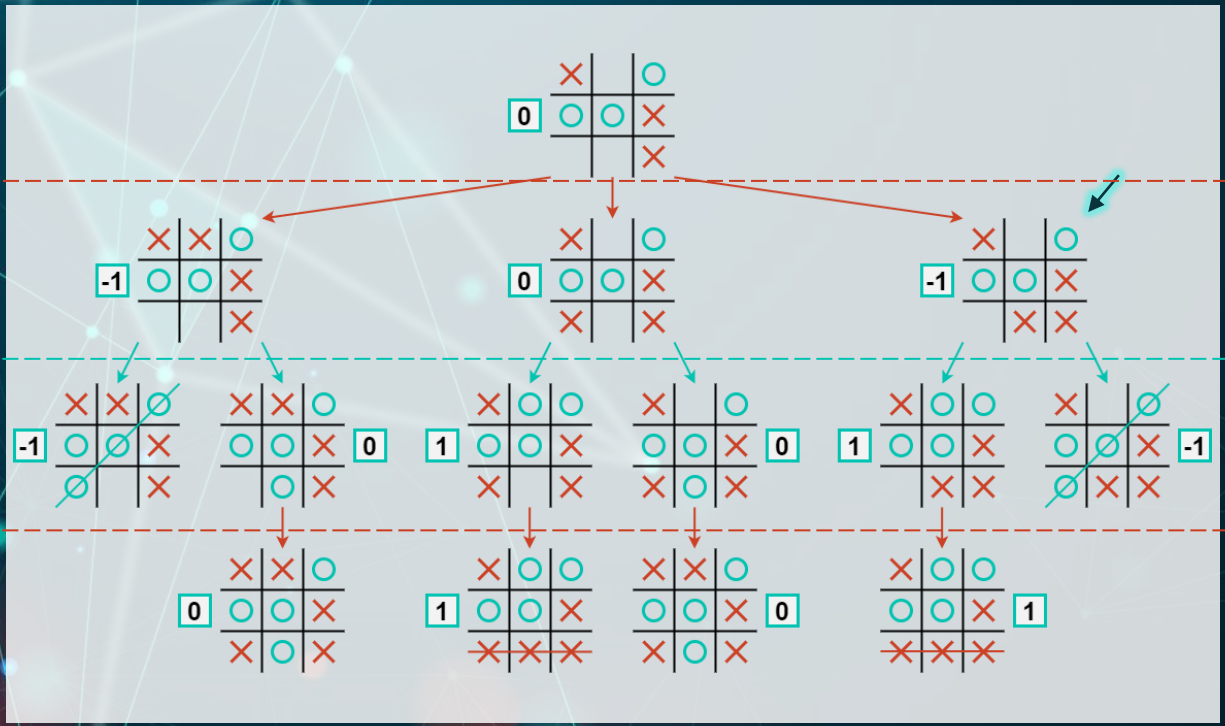
Zadatak 1 – Rešenje



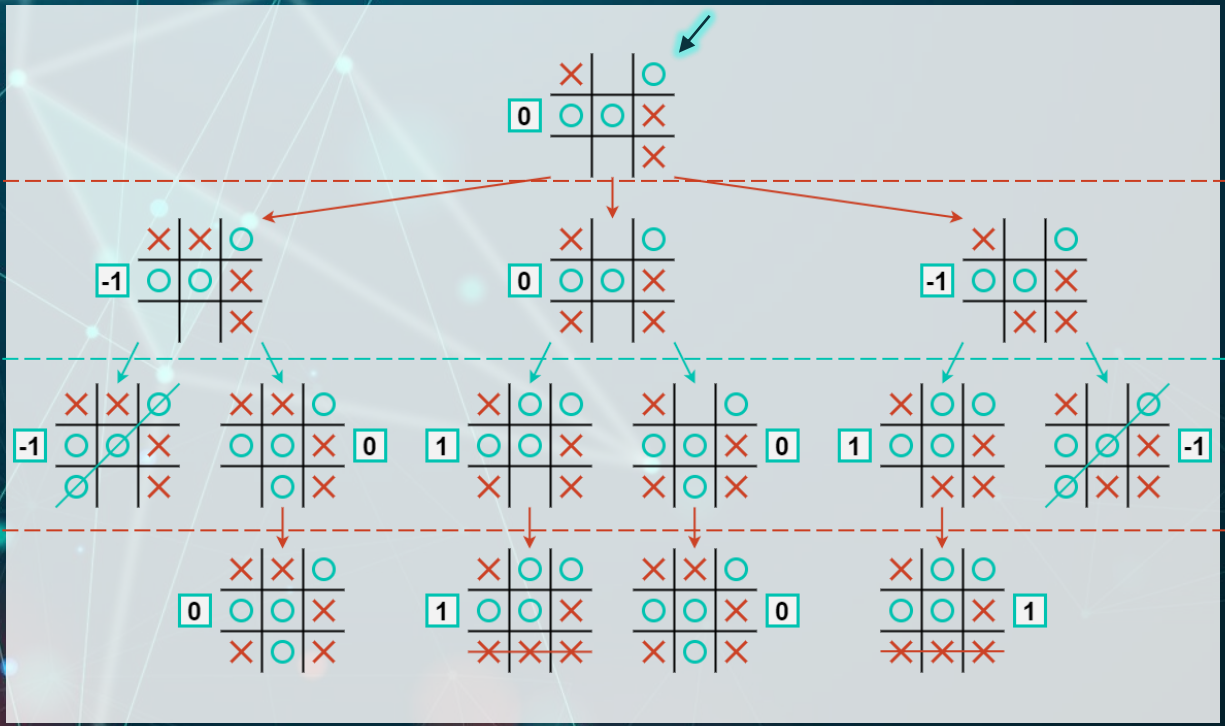
Zadatak 1 – Rešenje



Zadatak 1 – Rešenje

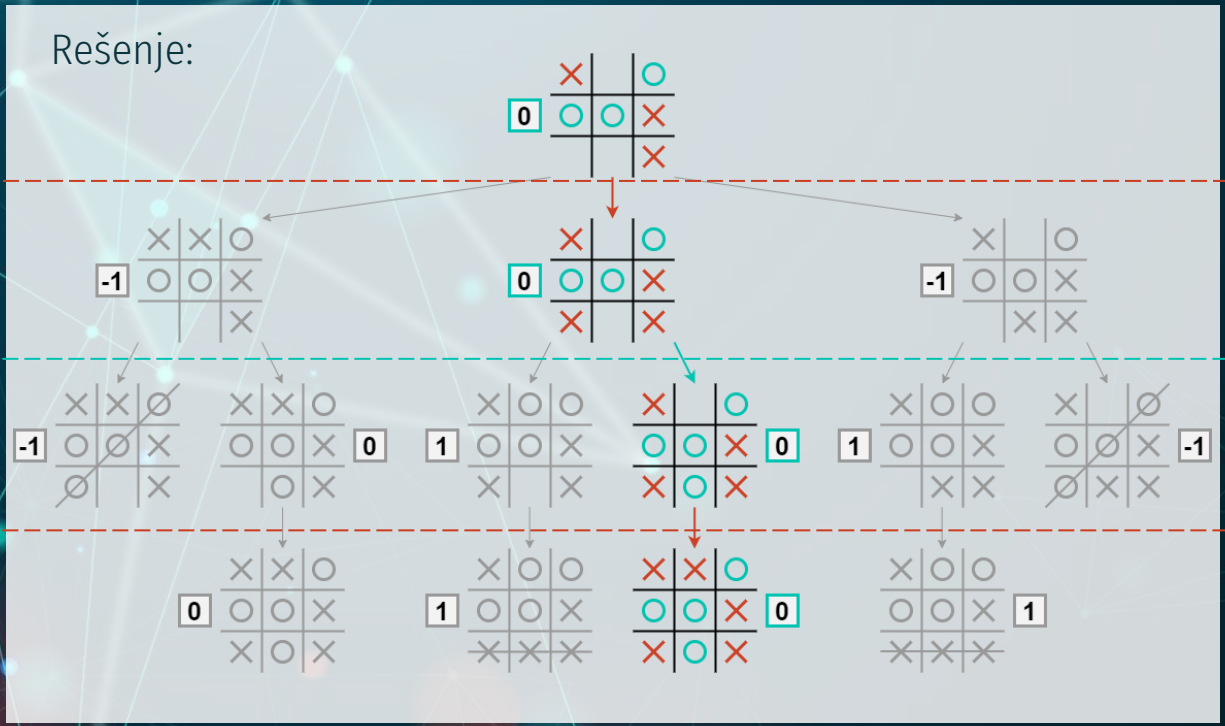


Zadatak 1 – Rešenje



Zadatak 1 – Rešenje

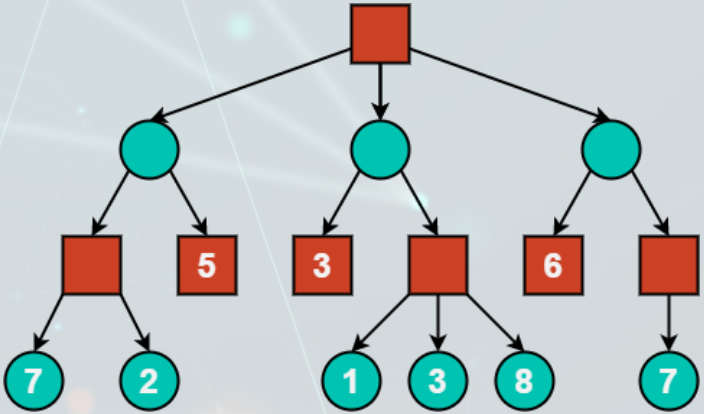
Rešenje:



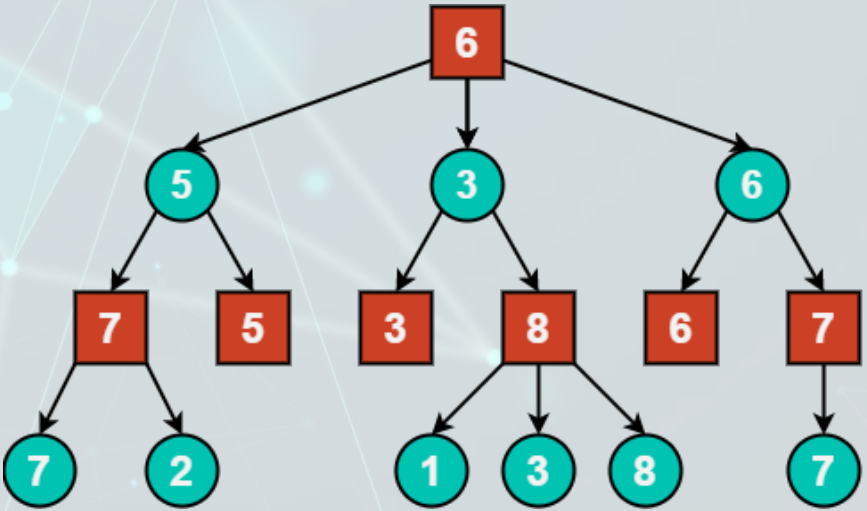
Zadatak za samostalnu vežbu – Minimax

★☆☆☆☆

Dato je kompletno stablo igre sa naznačenim dobicima za prvog igrača. Primenom minimax algoritma odrediti funkciju procene za svaki čvor, konačan ishod igre i sekvencu poteza koja će biti odigrana do kraja ukoliko su oba igrača racionalna.



Zadatak za samostalnu vežbu – Rešenje

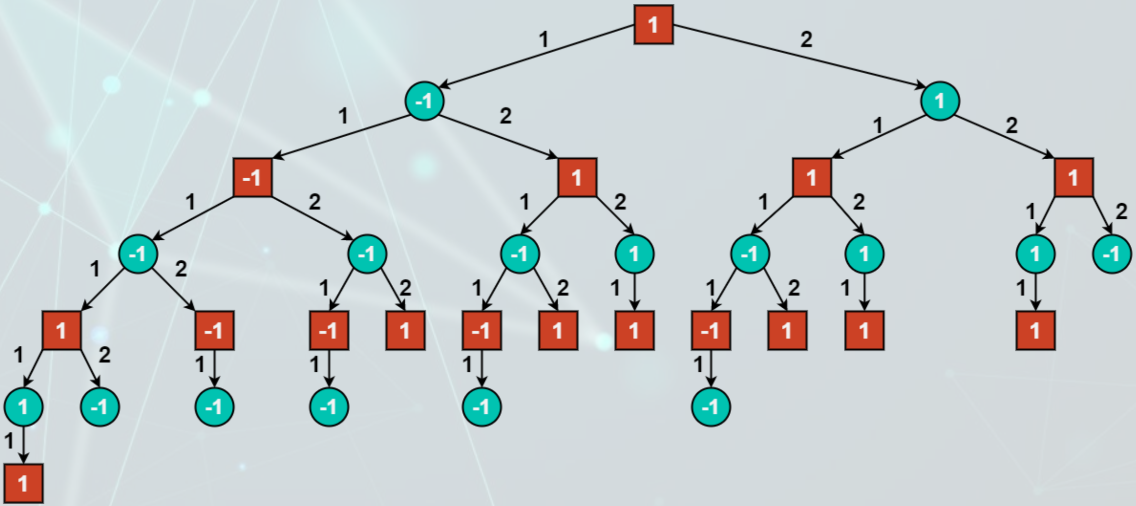


Zadatak za samostalnu vežbu – Novčići

★★☆☆☆

Arturovi vitezovi Gavejn i Persival ne mogu da idu u lov, jer pada jaka kiša. Odlučili su da ostanu u zamku i uz vino igraju njihovu omiljenu igru – **Novčići**. Pravila igre su jednostavna: na stolu stoji 6 novčića. Igrači igraju naizmenično i svaki igrač može da uzme 1 ili 2 novčića kada je njegov potez na redu. Igrač koji poslednji vrši uzimanje je izgubio ukoliko pri tom uzimanju na stolu nije ostao nijedan novčić. Koristeći minimax algoritam odrediti ishod igre ukoliko oba igrača igraju racionalno, a igru započinje Persival.

Zadatak za samostalnu vežbu - Rešenje



Zadatak 2 – XO (Progresivno produbljivanje)

★★☆☆☆

Data je jedna pozicija u jednoj partiji igre iks-oks. Predložiti funkciju procene koja će biti korišćena za evaluaciju čvorova na dubini d i primenom progresivnog produbljivanja koristeći **minimax** algoritam odrediti vrednost funkcije procene za svaki čvor i odluku igrača X ukoliko je moguće razviti čvorove na dubini 1 i 2, respektivno. Oba igrača su racionalna i na potezu je igrač X. Naslednike čvora razvijati dodajući znakove na slobodna polja po redovima.

Zadatak 2 – XO (Progresivno produbljivanje)

★★☆☆☆

Kako oceniti stanje koje nije terminalno (ne označava kraj igre), a nalazi se na dubini d?

Ukoliko se terminalni čvorovi označavaju sa vrednošću A za pobedu igrača MAX, odnosno B za pobedu MIN onda funkcija procene ne bi trebalo da vraća vrednosti van ovog opsega [B, A]. Ako to nije slučaj, onda bi igrač MAX potencijalno izabrao odluku modelovanu granom ka čvoru sa vrednošću funkcije procene većom od A umesto potencijalno pobedničkog poteza sa cenom A.

U ovoj igri na tabli 3x3 postoje 3 horizontalne, 3 vertikalne i 2 dijagonalne linije na kojima igrač može ostvariti pobedu. Kvalitet neke pozicije može se odrediti funkcijom koja zavisi od broja linija na kojima igrač može ostvariti pobedu i broja linija na kojima to može uraditi njegov protivnik.

Zadatak 2 – XO (Progresivno produbljivanje)

★★☆☆☆

Kako oceniti stanje koje nije terminalno (ne označava kraj igre), a nalazi se na dubini d?

S1 = $\sum_{x=1}^{x=2} a * 10 \text{ ^ } (x - 1)$ – *a* linija na kojima igrač ima *x* oznaka

S2 = $\sum_{y=1}^{y=2} b * 10 \text{ ^ } y$ – *b* linija na kojima protivnik ima *y* oznaka

Kad je MAX na potezu, vrednost funkcije procene naslednika:

node_evaluation(node) = +S1 - S2

Kad je MIN na potezu, vrednost funkcije procene naslednika:

node_evaluation(node) = -S1 + S2

Terminalno stanje se može oceniti vrednošću +1000 za igrača MAX, odnosno -1000 za igrača MIN.

ITERATIVE DEEPENING MINIMAX ALGORITAM

```
def id_minimax(node, player, depth, MAX_DEPTH):
    if depth == MAX_DEPTH or is_terminal_node(node):
        return node_evaluation(node)

    if player == Player.MAX:
        score = -math.inf
        for succ in node.successors():
            score = max(score, id_minimax(succ, Player.MIN, depth+1, MAX_DEPTH))
        return score
    else:
        score = +math.inf
        for succ in node.successors():
            score = min(score, id_minimax(succ, Player.MAX, depth+1, MAX_DEPTH))
        return score
```

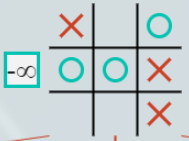
Zadatak 2 – Rešenje

S1 = $\sum_{x=1}^{x=2} a * 10 \text{ ^ } (x - 1)$

S2 = $\sum_{y=1}^{y=2} b * 10 \text{ ^ } y$

node_evaluation(node) = +S1 - S2

Dubina 1 - Evaluacija



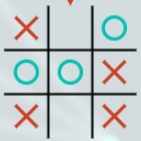
MAX

MIN

S1 = 1 * 10^0 = 1

S2 = 1 * 10^2 = 100

S1 – **S2** = -99



S1 = 1 * 10^1 = 10

S2 = 1 * 10^1 = 10

S1 – **S2** = 0

S1 = 1 * 10^1 = 10

S2 = 1 * 10^2 = 100

S1 – **S2** = -90

Zadatak 2 – Rešenje

$S1 = \sum_{x=1}^{x=2} a * 10 \text{ ^ } (x - 1)$

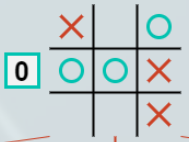
$S2 = \sum_{y=1}^{y=2} b * 10 \text{ ^ } y$

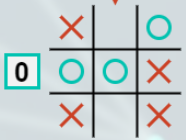
node_evaluation(node) = +S1 - S2

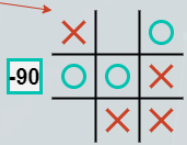
Dubina 1 - Minimax

MAX

MIN







S1 = 1 * 10^0 = 1

S2 = 1 * 10^2 = 100

S1 – **S2** = -99

S1 = 1 * 10^1 = 10

S2 = 1 * 10^1 = 10

S1 – **S2** = 0

S1 = 1 * 10^1 = 10

S2 = 1 * 10^2 = 100

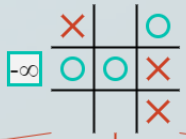
S1 – **S2** = -90

Zadatak 2 – Rešenje

S1 = $\sum_{x=1}^{x=2} a * 10 \text{ ^ } (x - 1)$

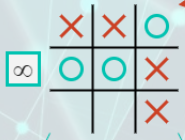
S2 = $\sum_{y=1}^{y=2} b * 10 \text{ ^ } y$

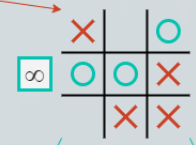
node_evaluation(node) = +S2 – S1



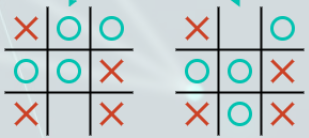
Dubina 2 - Evaluacija

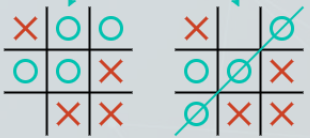
MAX





MIN





MAX

S1 = 1 * 10^1 = 10

S2 = 0

S2 – **S1** = -10

S1 = 1 * 10^1 = 10

S2 = 1 * 10^2 = 100

S2 – **S1** = 90

S1 = 1 * 10^1 = 10

S2 = 0

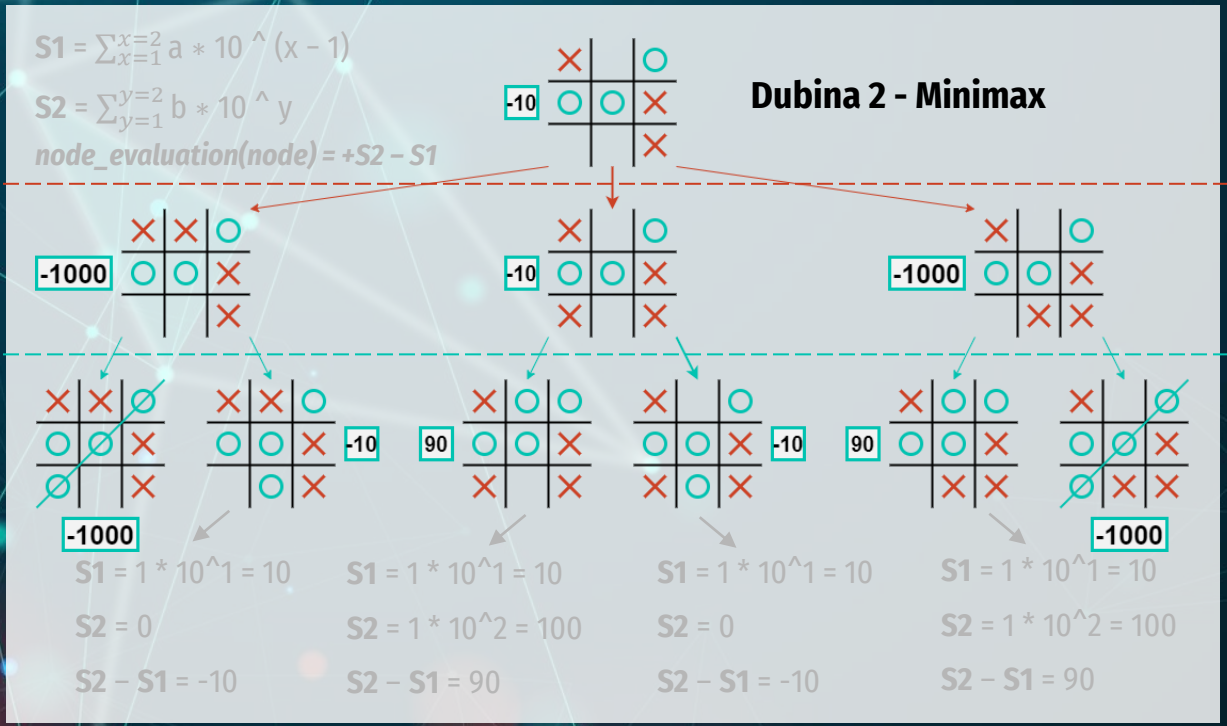
S2 – **S1** = -10

S1 = 1 * 10^1 = 10

S2 = 1 * 10^2 = 100

S2 – **S1** = 90

Zadatak 2 – Rešenje



Zadatak 3 – XO (Alfa-Beta odsecanje)

★★★☆☆

Data je jedna pozicija u jednoj partiji igre iks-oks. Primenom **minimax** algoritma sa **alfa-beta odsecanjem** odrediti funkciju procene za svaki čvor, konačan ishod igre i sekvencu poteza koja će biti odigrana do kraja ukoliko su oba igrača racionalna. Vrednost funkcije procene terminalne pozicije označiti sa +1 ukoliko je pobednik X, -1 ukoliko je pobednik O i 0 ukoliko je ishod igre nerešen. Na potezu je igrač X. Naslednike čvora razvijati dodajući znakove na slobodna polja po kolonama.

Alfa-Beta odsecanje IDEJA

Ideja **alfa-beta odsecanja** je da unapredi i ubrza minimax algoritam tako što će odsecanjem određenih podstabala kompletnog stabla igre smanjiti broj čvorova za koje se vrši evaluacija funkcije procene.

Za poziciju A i njene naslednike B i C, nakon evaluacije funkcije procene pozicije B, algoritam prestaje da evaluira funkciju procene za poziciju C ukoliko se u nekom trenutku evaluacije funkcije procene za poziciju C pokaže da je ta pozicija i u najboljem slučaju gora od pozicije B. U tom slučaju je racionalan igrač nikad neće izabrati sa obzirom da mu je pozicija B bolja.

Algoritam vraća isti rezultat kao i običan minimax algoritam, ali odseca grane koje vode do pozicija koje ne utiču na konačnu odluku izbora najbolje pozicije.

Alfa-Beta odsecanje IDEJA

Algoritam održava dve vrednosti:

- **alfa** – predstavlja minimalnu vrednost koja je garantovana igraču MAX. Igrač MAX, ako je racionalan, ne može odigrati potez koji mu donosi vrednost manju od alfa. Cilj igrača MAX je da ostvari što veću vrednost za sebe.
- **beta** – predstavlja maksimalnu vrednost koja je garantovana igraču MIN. Igrač MIN, ako je racionalan, ne može odigrati potez koji mu donosi vrednost veću od beta. Cilj igrača MIN je da ostvari što manju vrednost za sebe.

MAX igrač ažurira vrednost alfa, dok MIN igrač ažurira vrednost beta. Ukoliko u nekom trenutku za neku poziciju važi **alfa >= beta**, tada igrač na potezu ne treba da razmatra ostale naslednike tekuće pozicije, jer mu protivnik takve poteze neće dozvoliti.

MINIMAX ALPHA-BETA ALGORITAM

```
def minimax_alpha_beta(node, player, alpha, beta):
    if is_terminal_node(node):
        return node_evaluation(node)

    if player == Player.MAX:
        score = -math.inf
        for succ in node.successors():
            score = max(score, minimax(succ, Player.MIN, alpha, beta))
            alpha = max(alpha, score)
            if alpha >= beta : break
        return score
    else:
        score = +math.inf
        for succ in node.successors():
            score = min(score, minimax(succ, Player.MAX, alpha, beta))
            beta = min(beta, score)
            if alpha >= beta : break
        return score
```

POBOLJŠANJE Alfa-Beta odsecanja

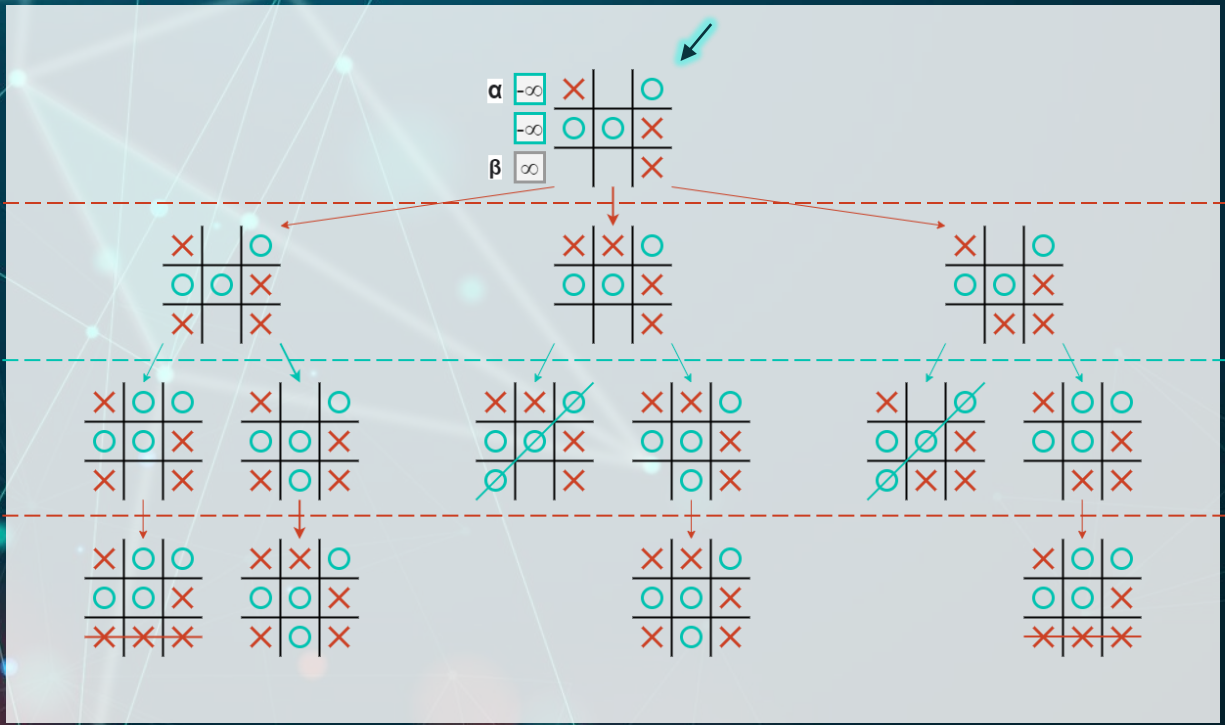
Algoritam alfa-beta odsecanja može poboljšati minimax algoritam odsecanjem određenih podstabala stabla igre koje predstavljaju nedostižne pozicije u igri za racionalne igrače.

Za stablo sa prosečnim faktorom grananja b i dubinom d minimax algoritam evaluira funkciju procene za b^d listova.

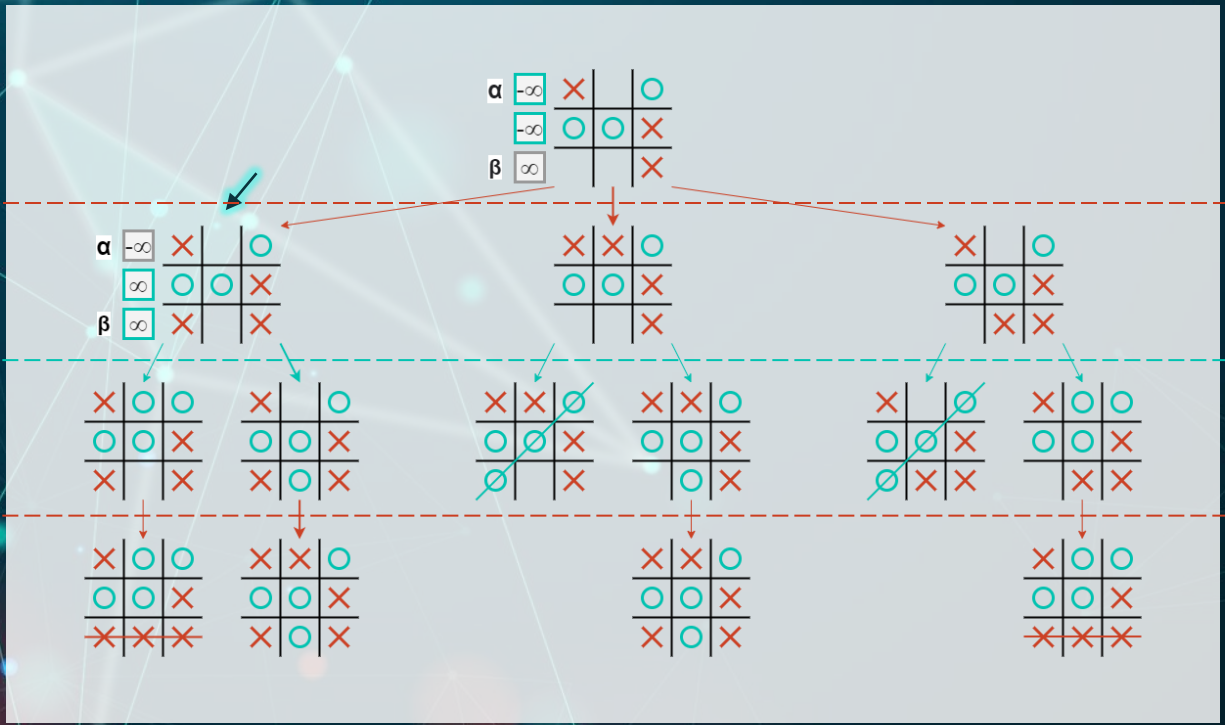
U najgorem slučaju alfa-beta poboljšanje radi evaluaciju istog broja listova.

U najboljem slučaju alfa-beta poboljšanje radi evaluaciju b^(d/2) listova. U takvom slučaju potrebno je, kao i u najgorem, razmatrati sve poteze prvog igrača (b poteza), ali za svaki potez prvog igrača neophodno je razmotriti samo 1 potez (i to najbolji) drugog igrača itd. To znači da se u najboljem slučaju za iste računarske resurse pretraga može vršiti za 2 puta veću dubinu.

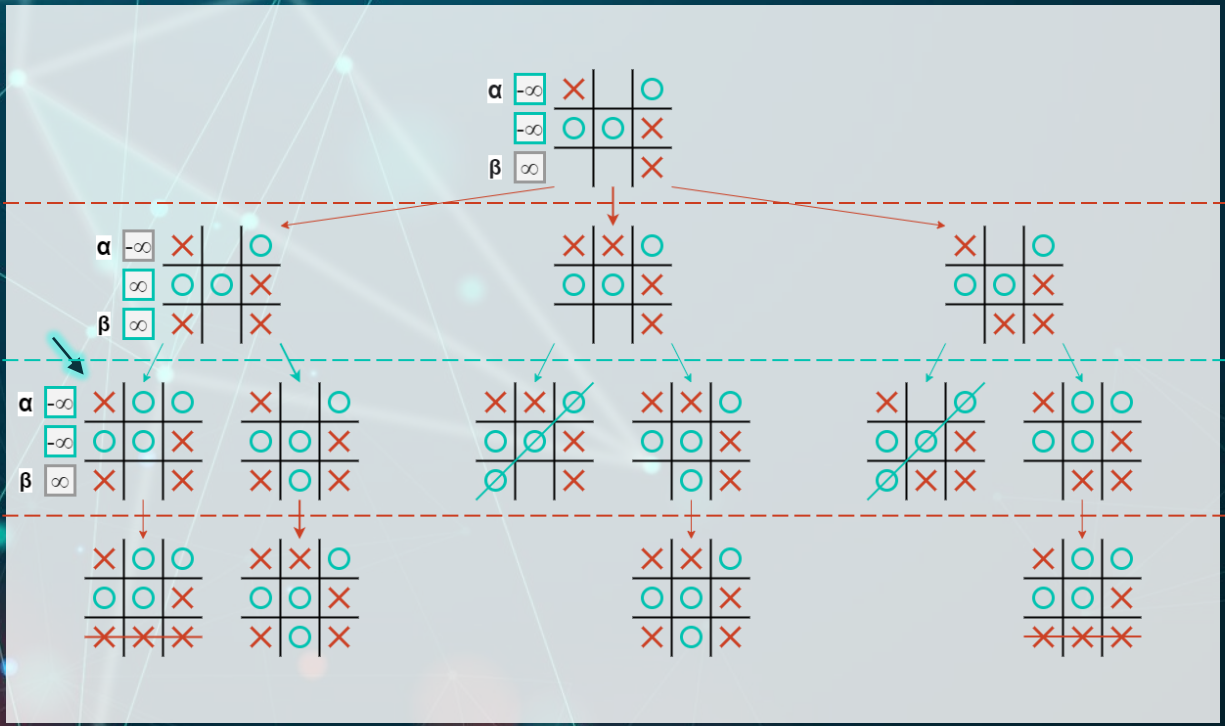
Zadatak 3 – Rešenje



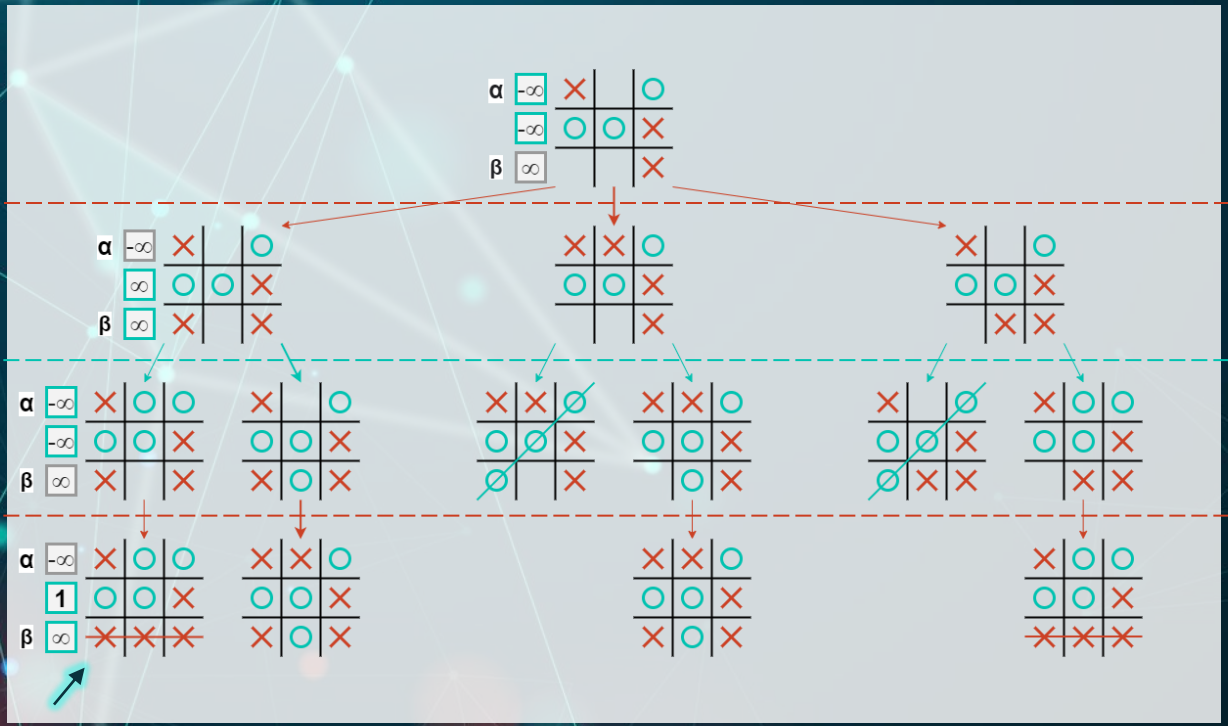
Zadatak 3 – Rešenje



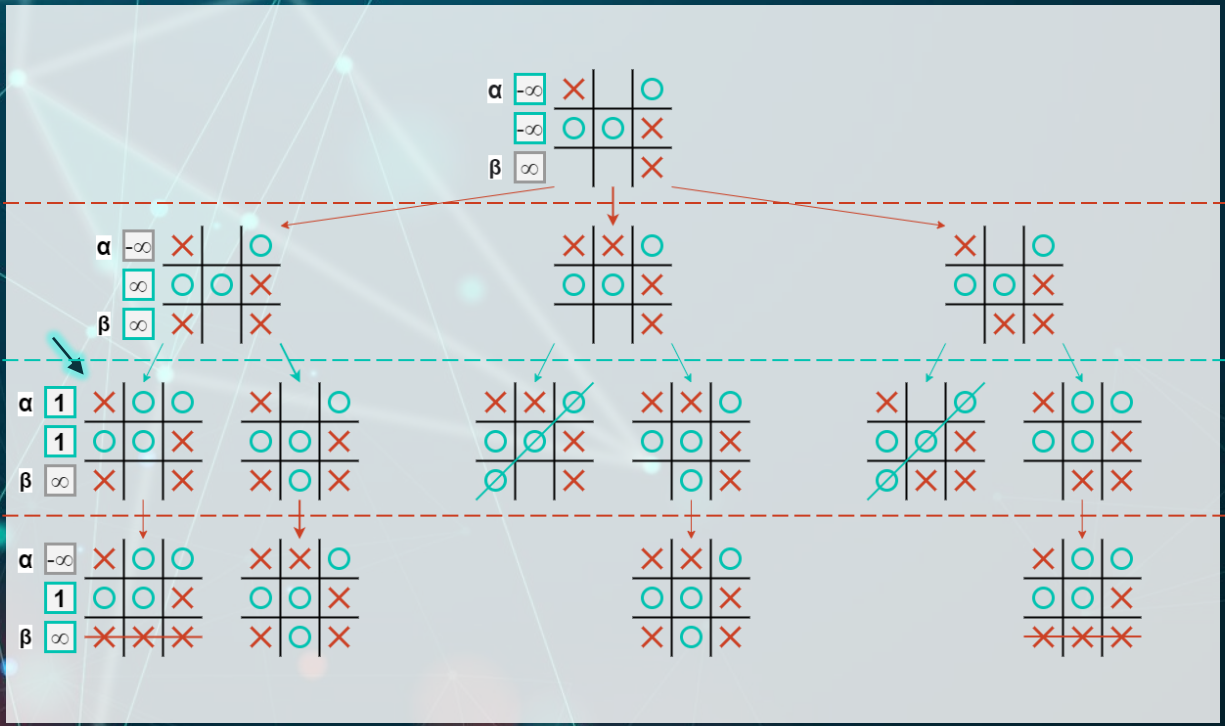
Zadatak 3 – Rešenje



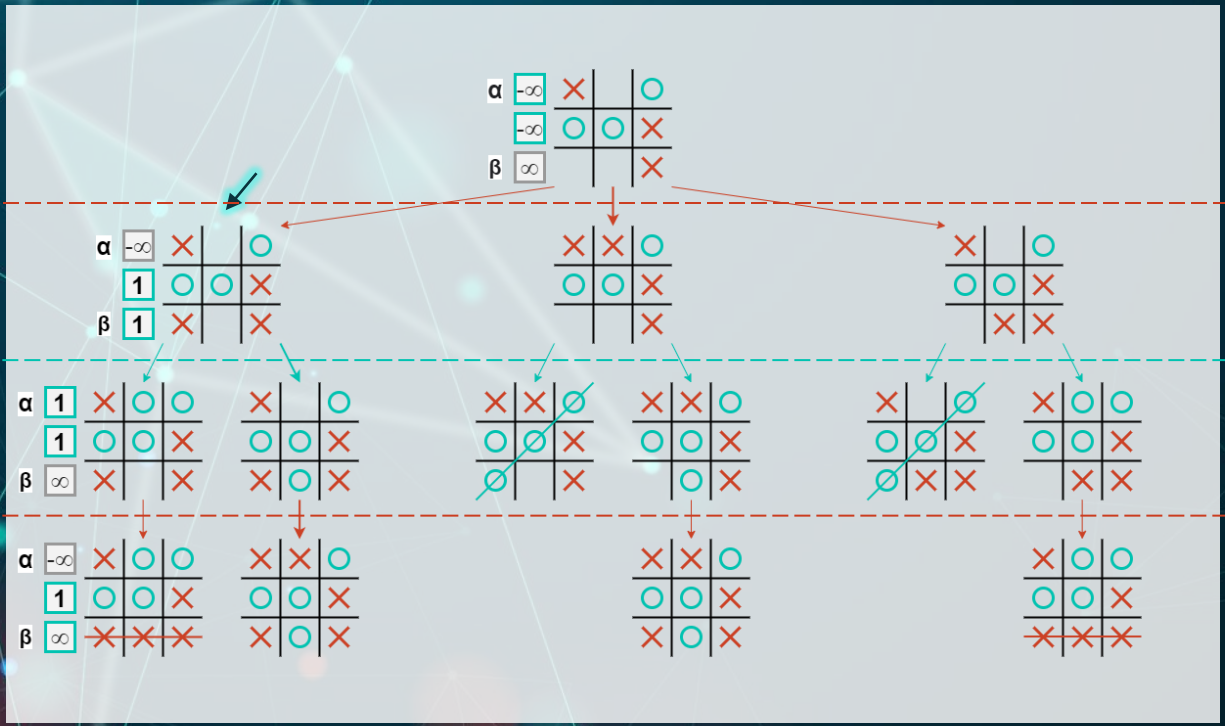
Zadatak 3 – Rešenje



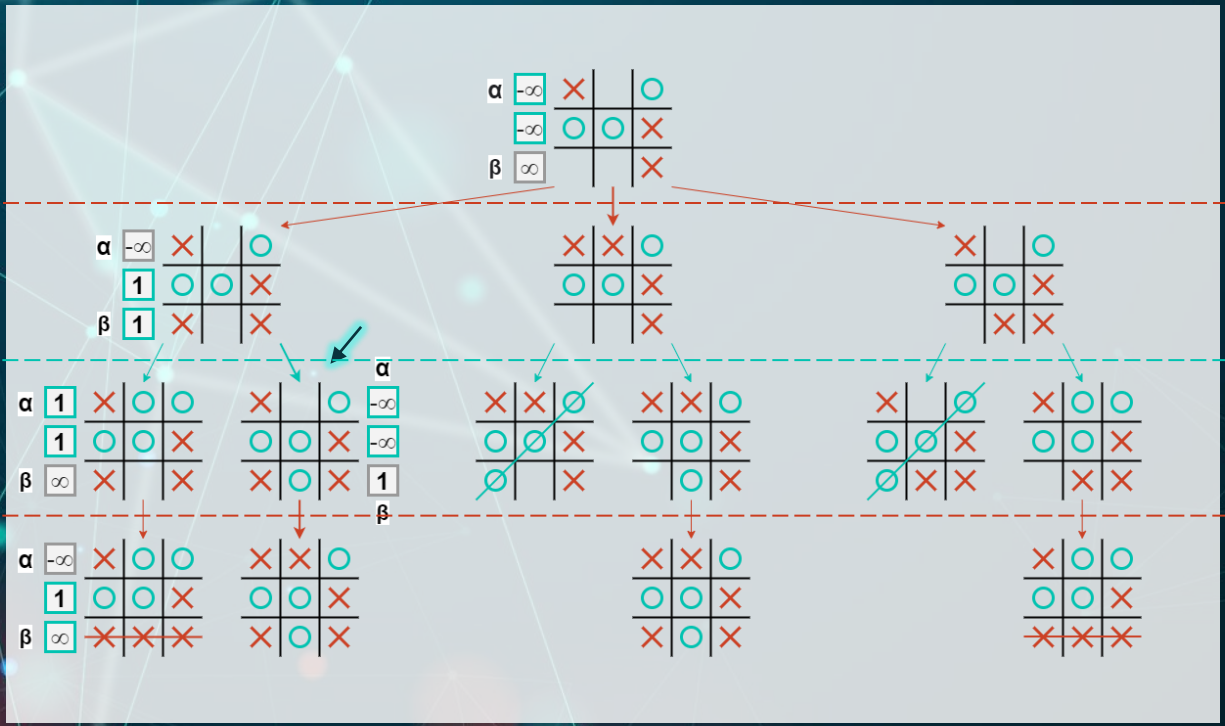
Zadatak 3 – Rešenje



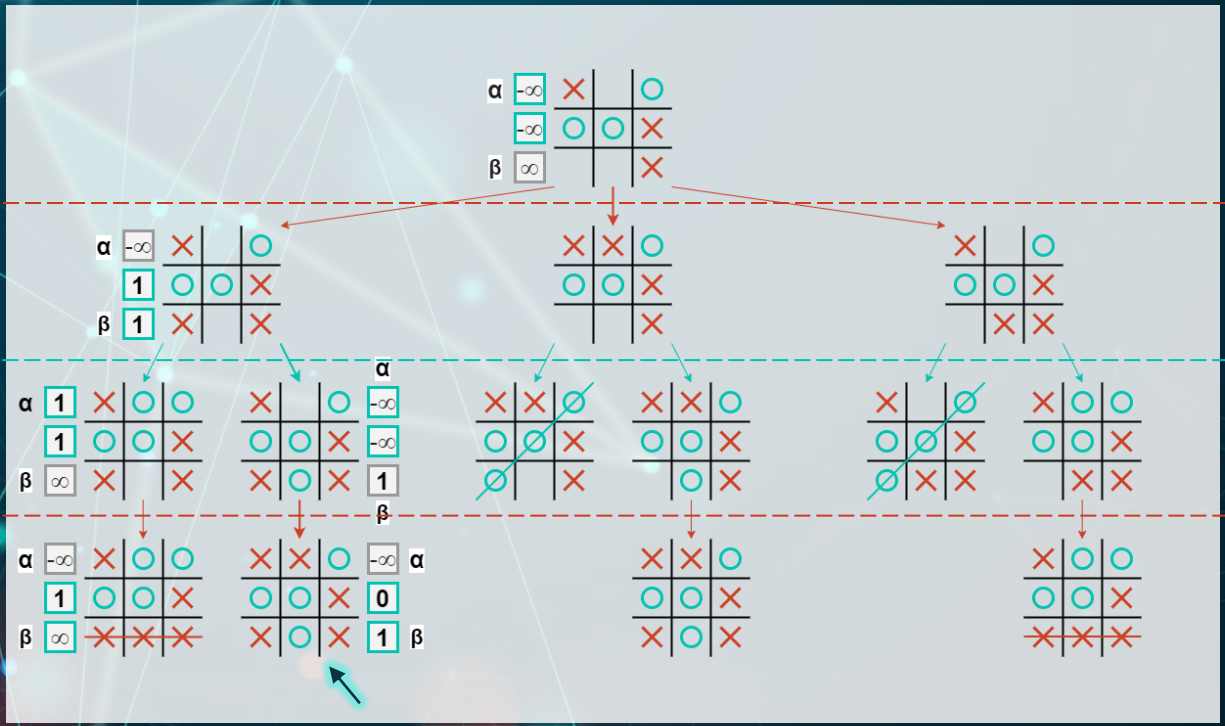
Zadatak 3 – Rešenje



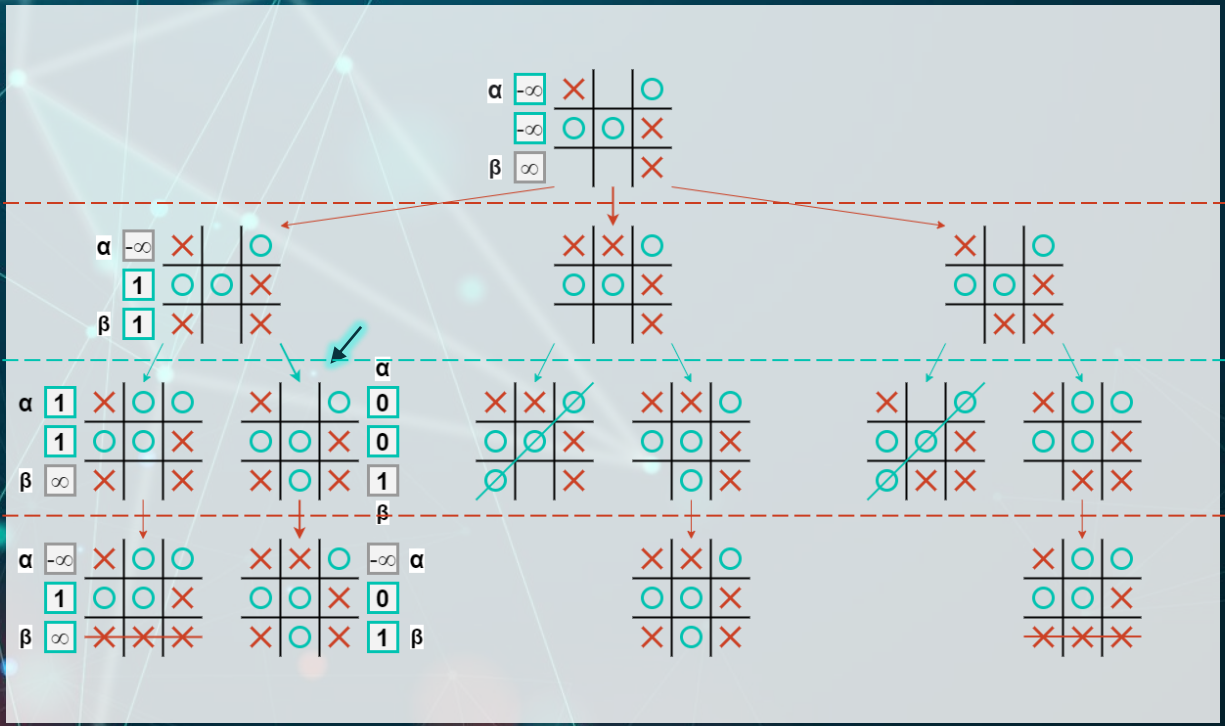
Zadatak 3 – Rešenje



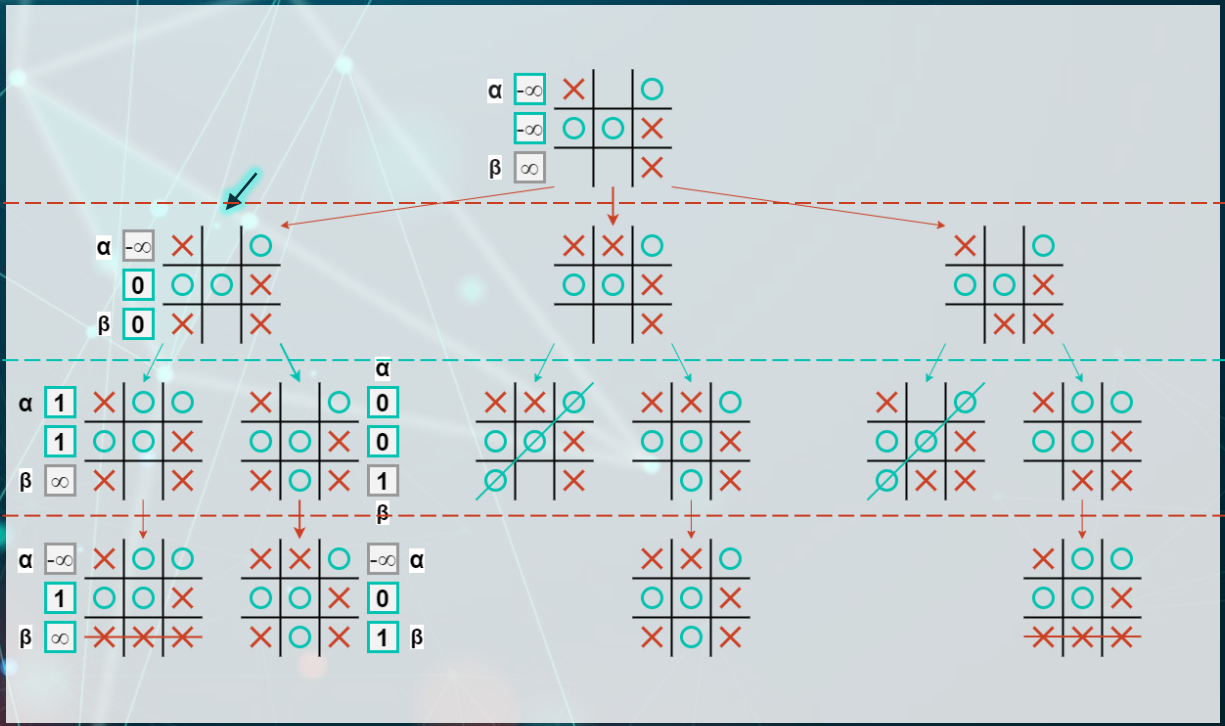
Zadatak 3 – Rešenje



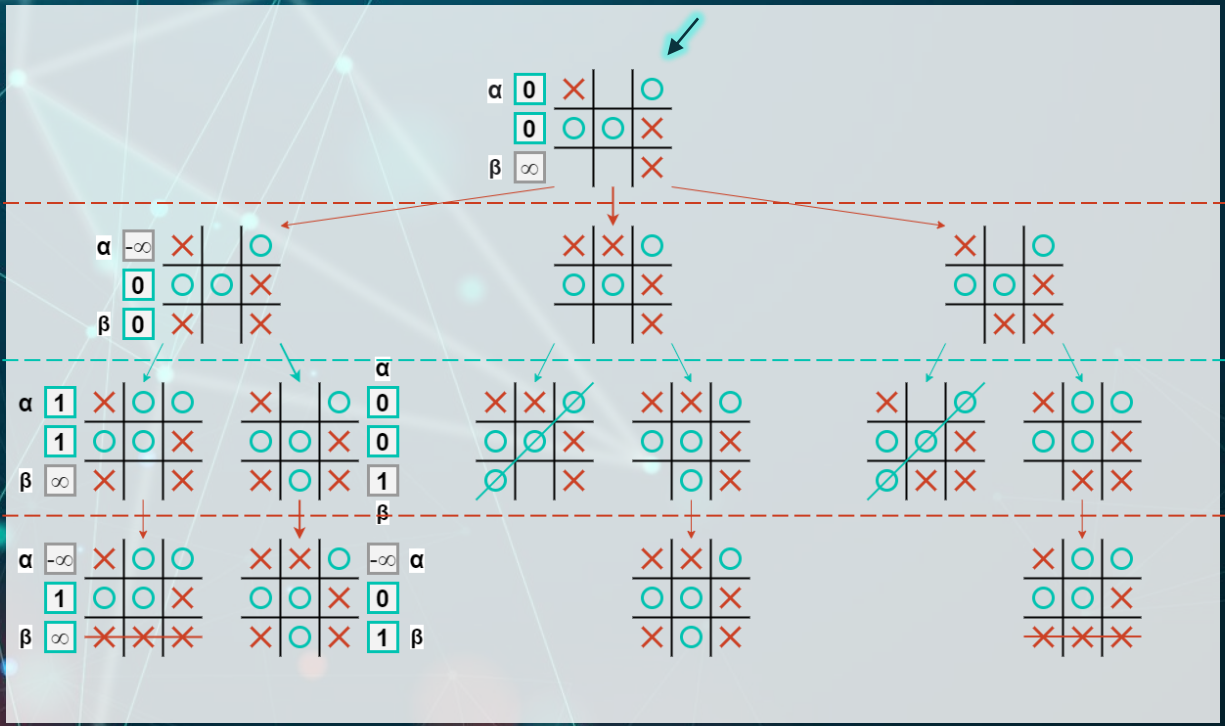
Zadatak 3 – Rešenje



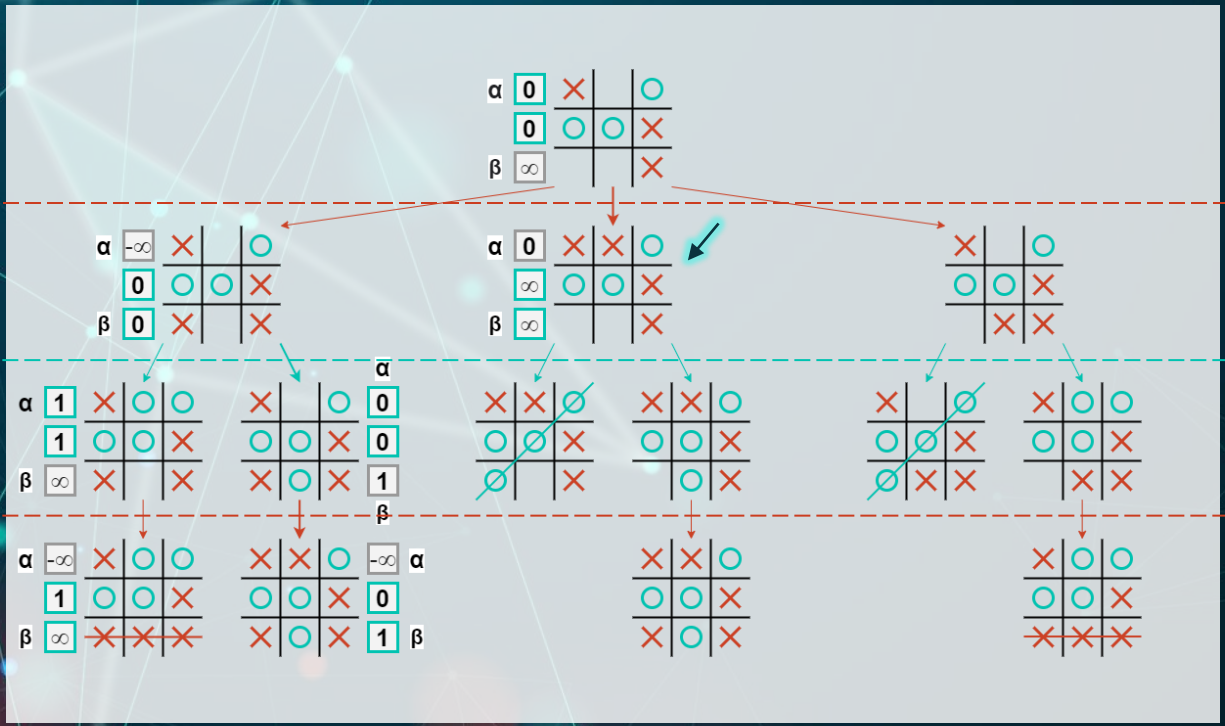
Zadatak 3 – Rešenje



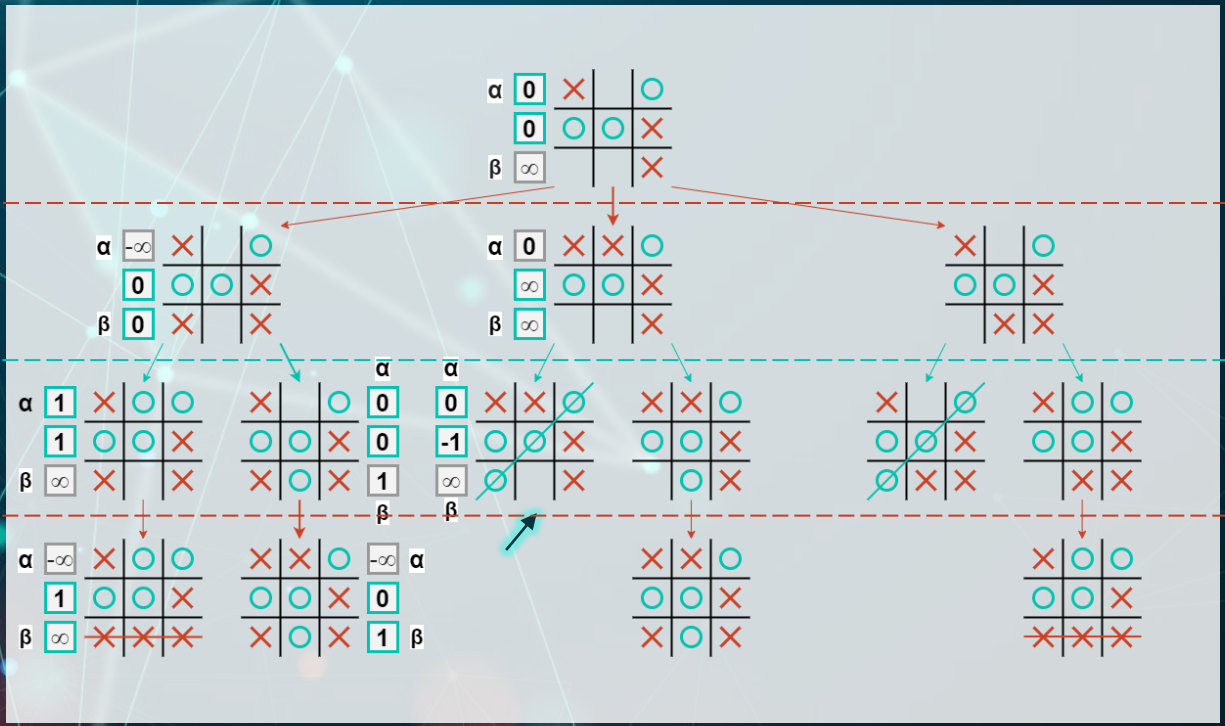
Zadatak 3 – Rešenje



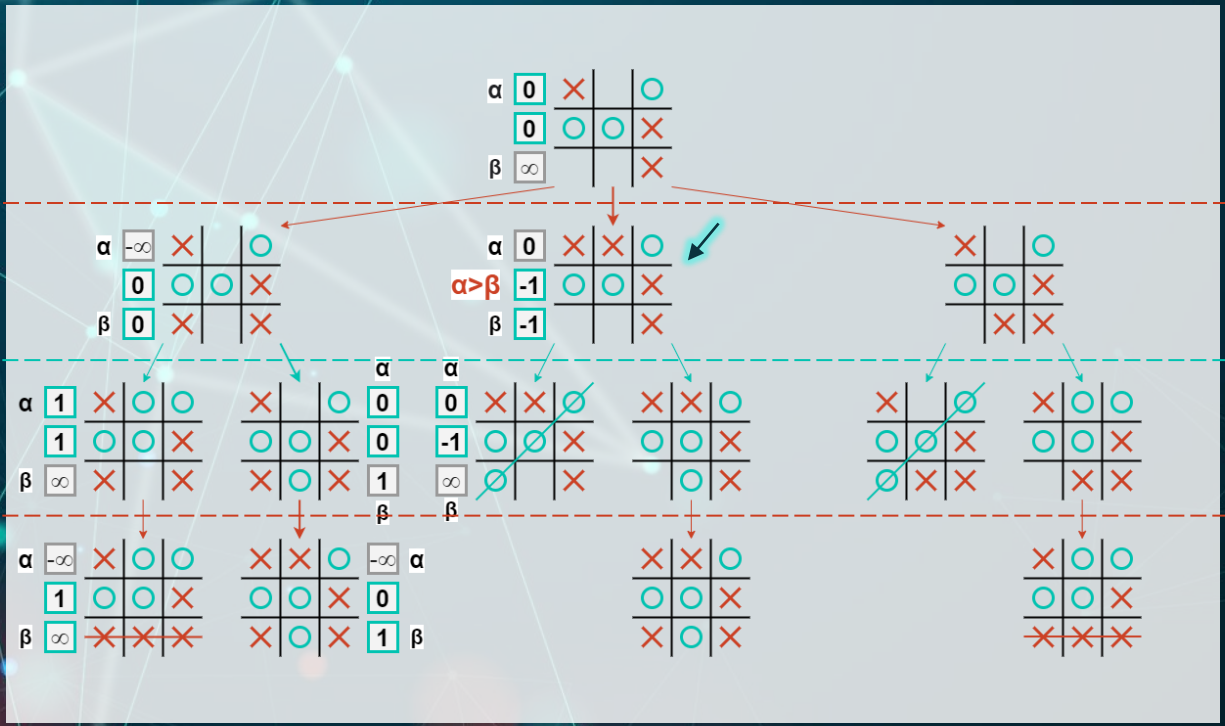
Zadatak 3 – Rešenje



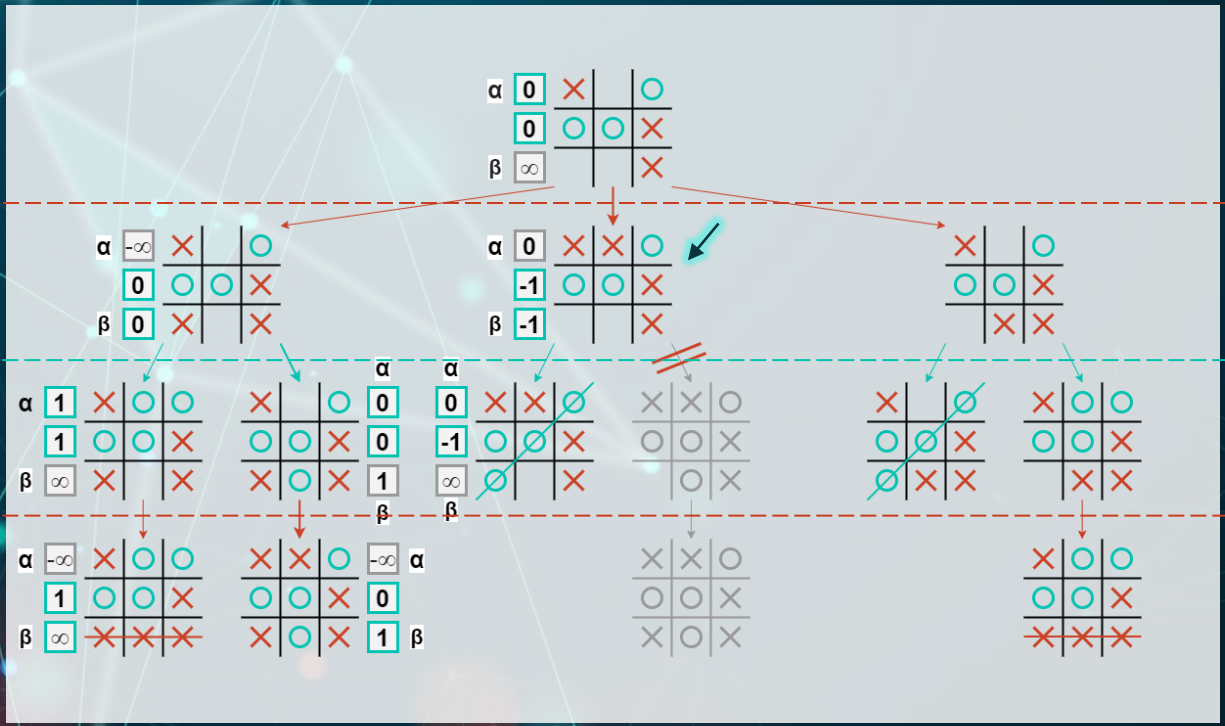
Zadatak 3 – Rešenje



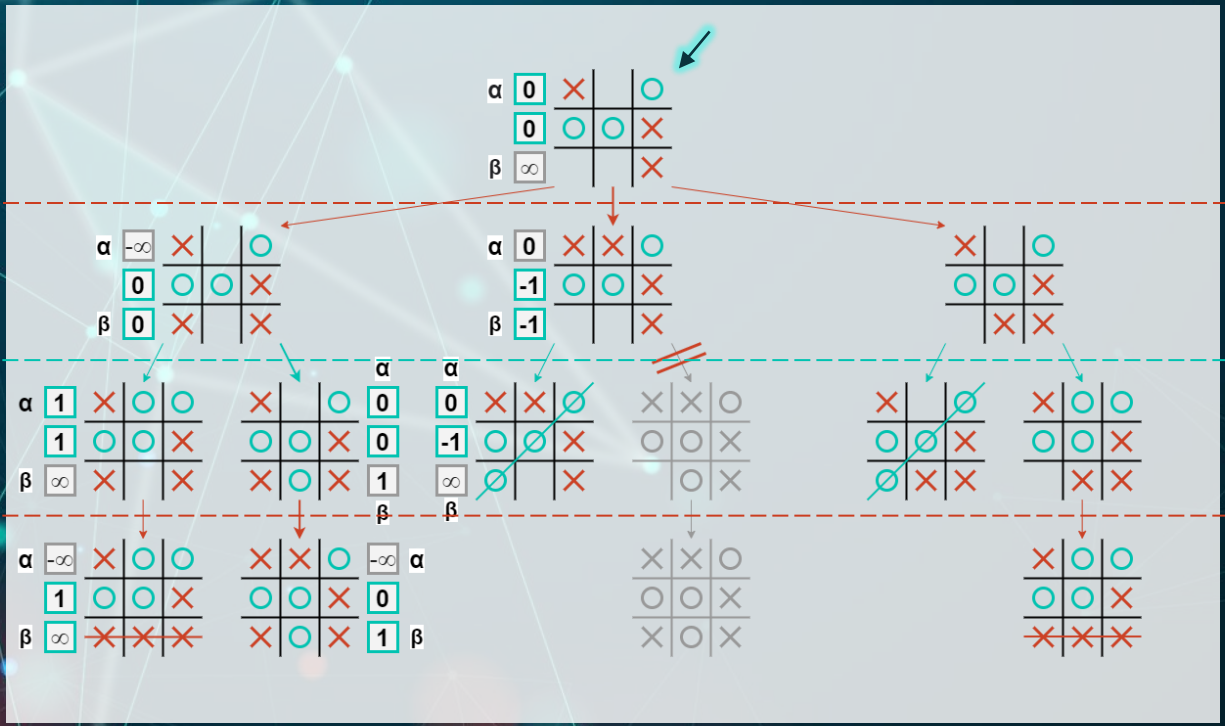
Zadatak 3 – Rešenje



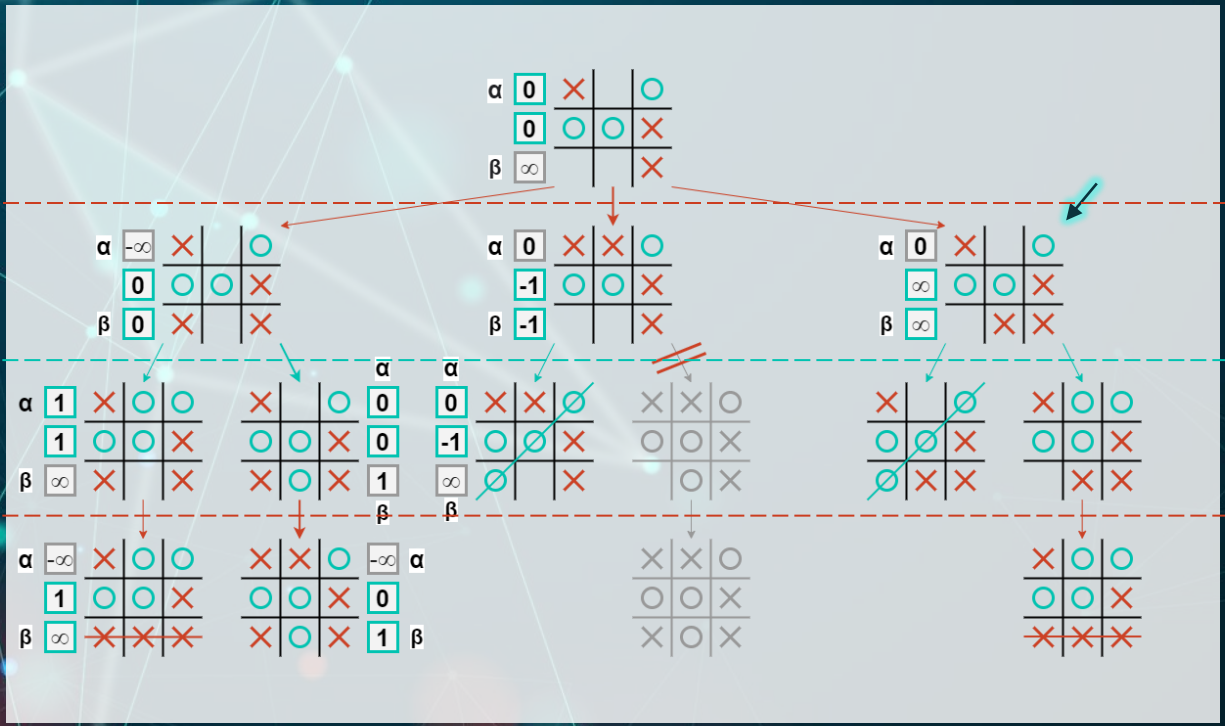
Zadatak 3 – Rešenje



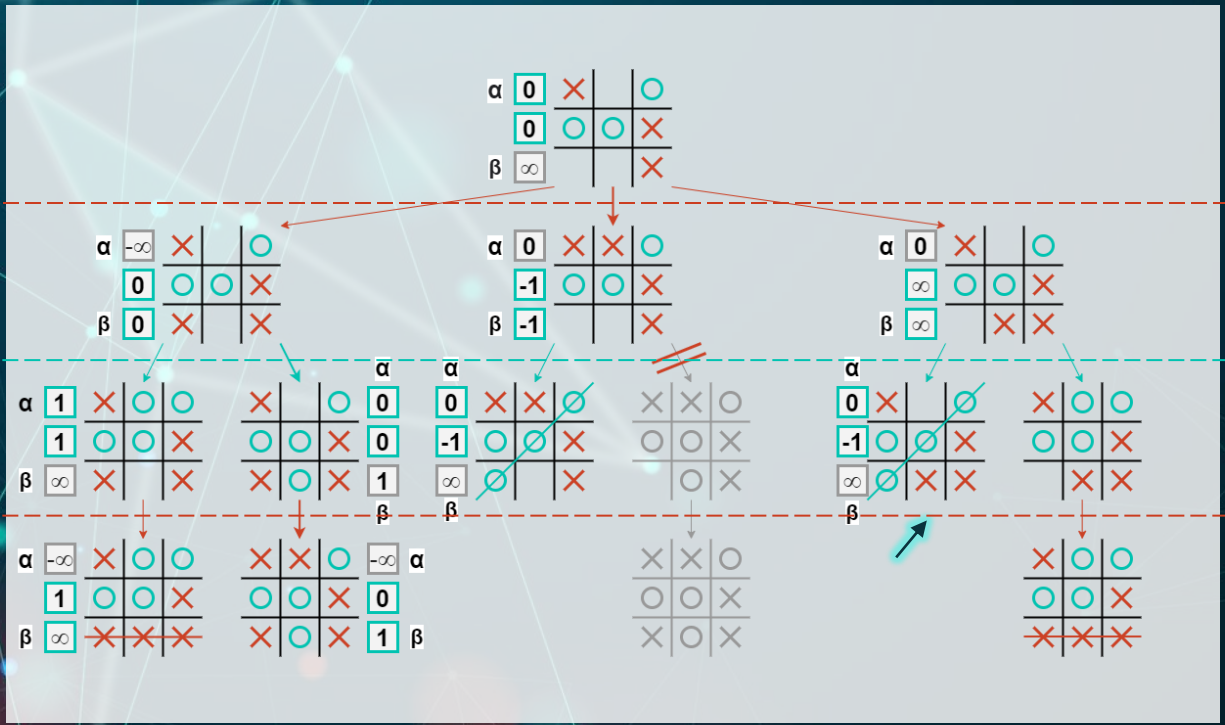
Zadatak 3 – Rešenje



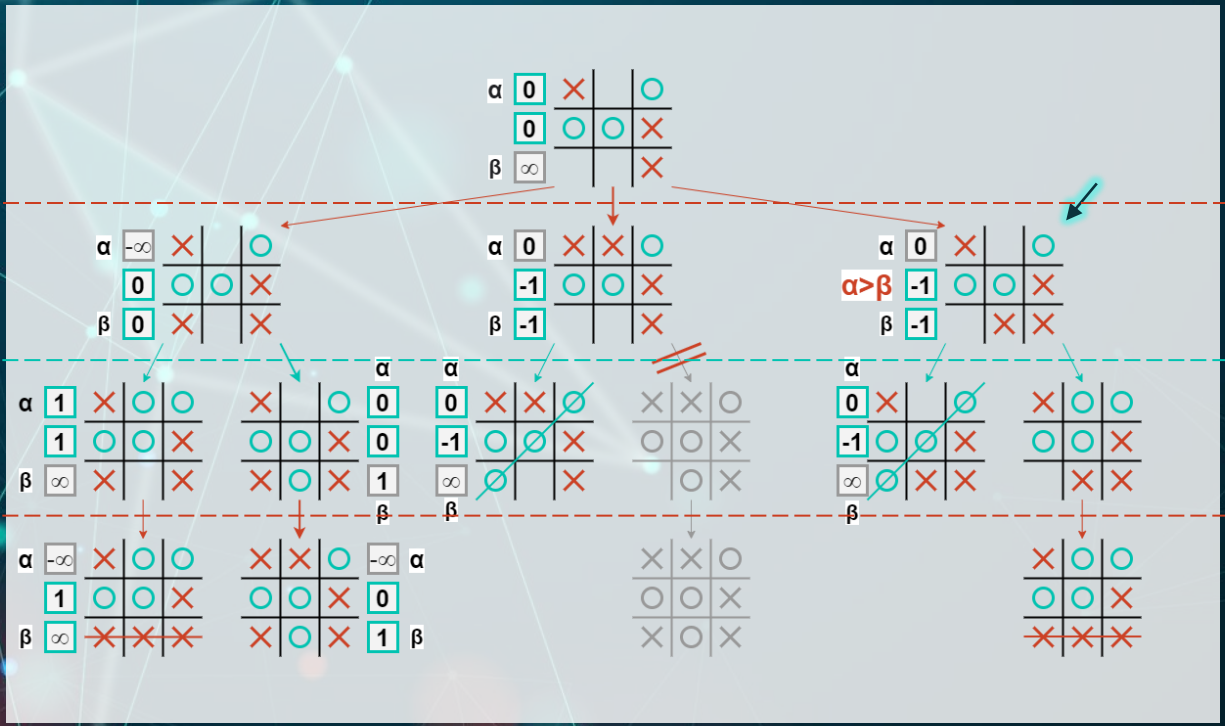
Zadatak 3 – Rešenje



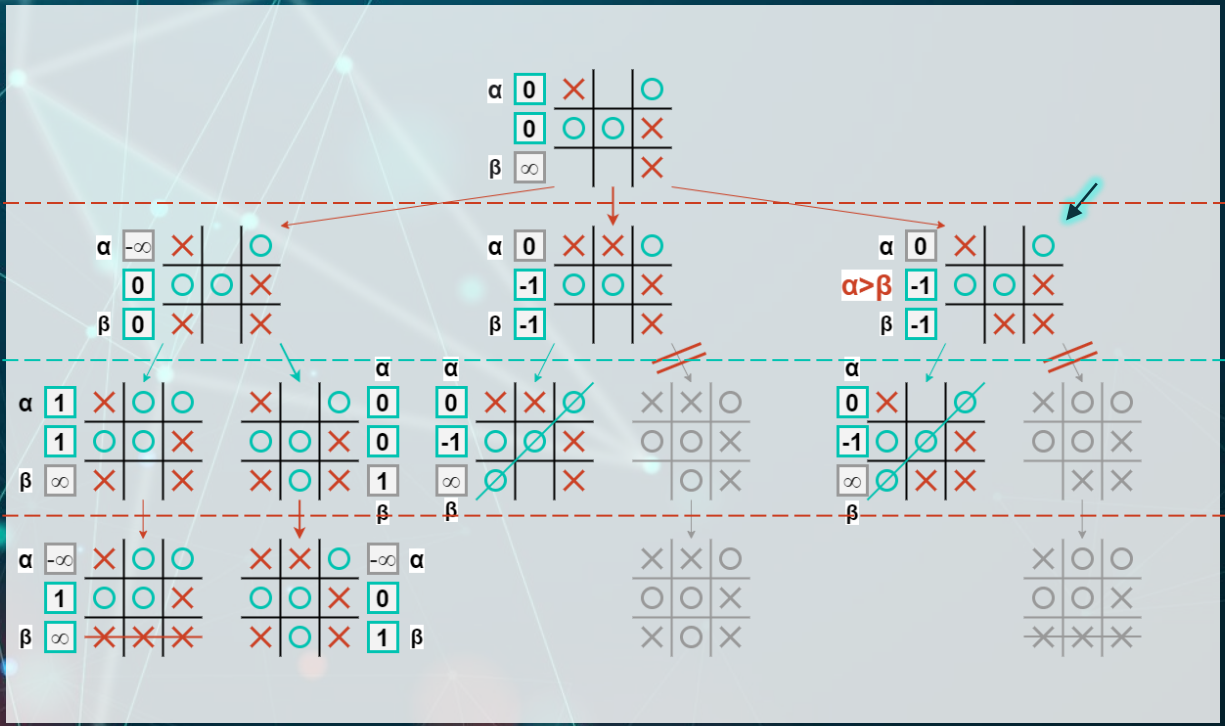
Zadatak 3 – Rešenje



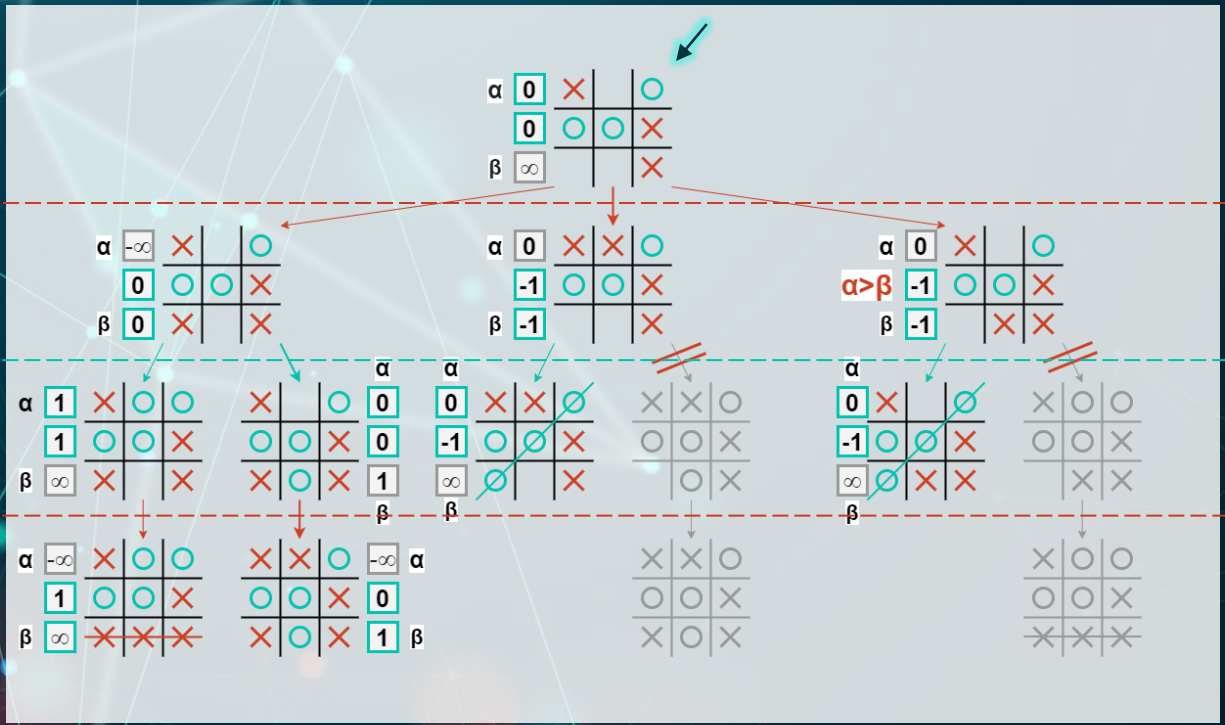
Zadatak 3 – Rešenje



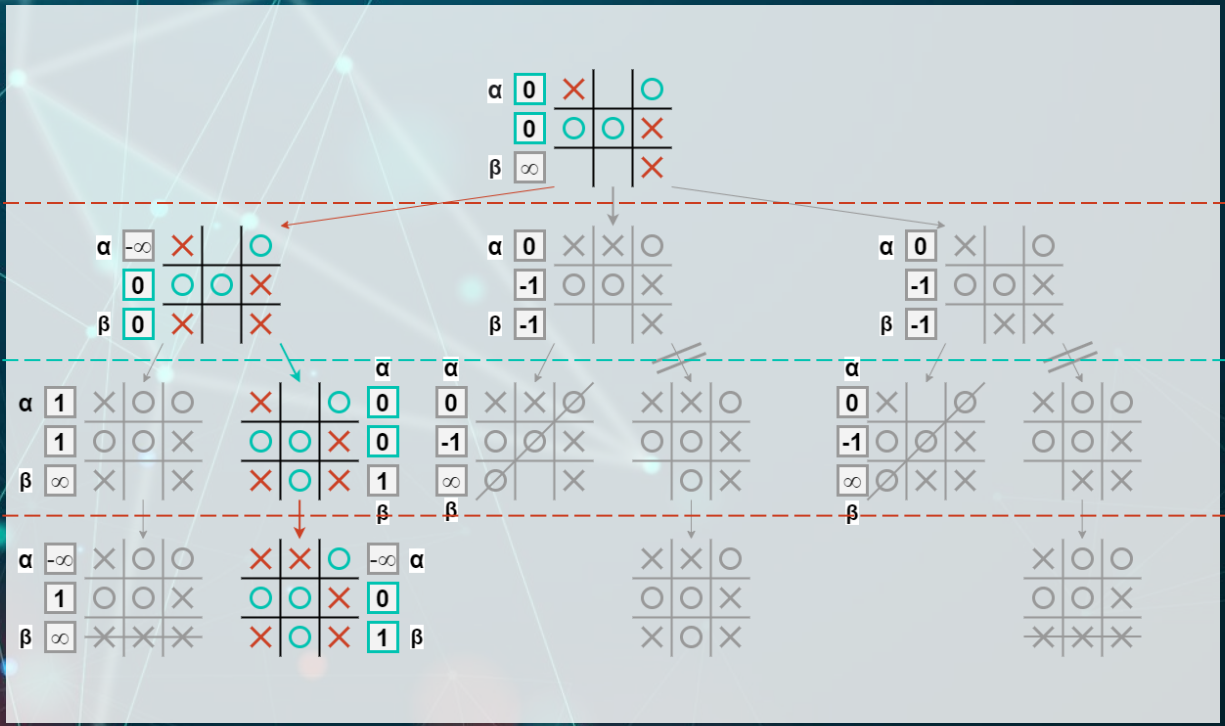
Zadatak 3 – Rešenje



Zadatak 3 – Rešenje



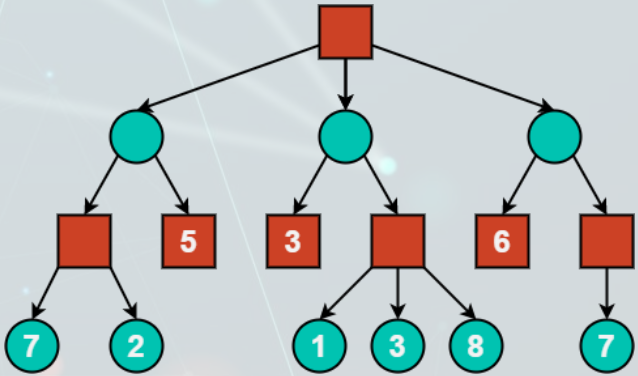
Zadatak 3 – Rešenje



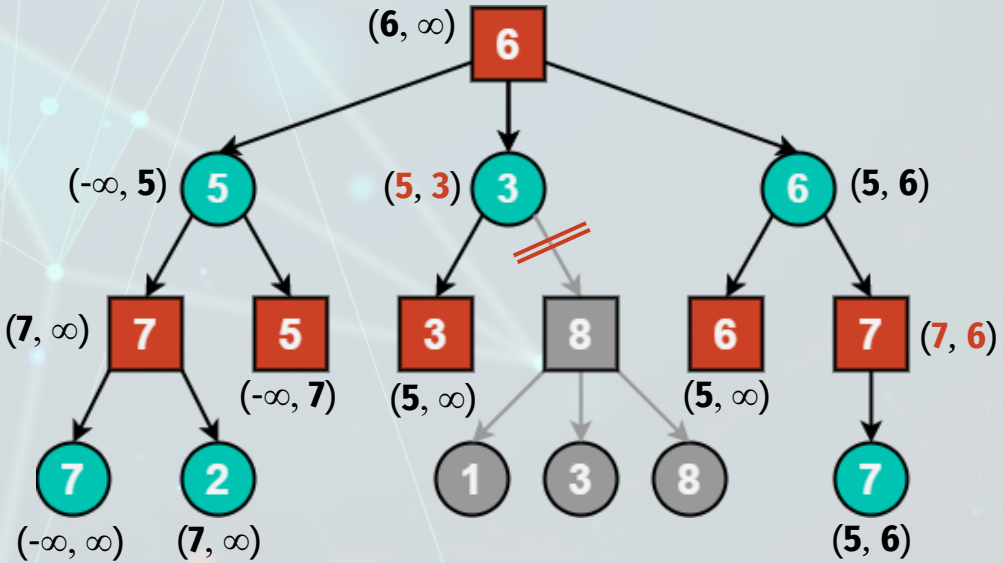
Zadatak za samostalnu vežbu – Alfa-Beta

★★☆☆☆

Dato je kompletno stablo igre sa naznačenim dobicima za prvog igrača. Primenom minimax algoritma sa alfa-beta odsecanjem odrediti funkciju procene za svaki čvor ukoliko su oba igrača racionalna. Da li je moguće na drugačiji način rasporediti poteze u stablu tako da broj odsecanja bude veći?



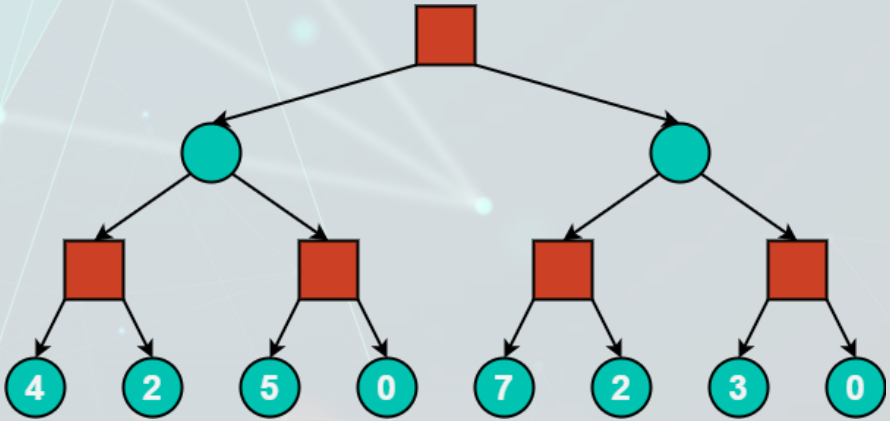
Zadatak za samostalnu vežbu – Rešenje



Zadatak 4 – Negamax

★★★☆☆

Dato je kompletno stablo igre sa naznačenim dobicima za prvog igrača. Primenom **negamax** algoritma odrediti funkciju procene za svaki čvor, konačan ishod igre i sekvencu poteza koja će biti odigrana do kraja ukoliko su oba igrača racionalna.



NEGAMAX IDEJA

Ideja **negamax** algoritma je da pojednostavi implementaciju minimax algoritma tako što neće razdvajati obradu na delove koje se tiču MAX i delove koji se tiču MIN igrača.

Za razliku od minimax algoritma gde MAX igrač vrši maksimizaciju dobitaka svih pozicija naslednika tekuće pozicije, a MIN igrač minimizaciju, u negamax algoritmu oba igrača vrše maksimizaciju dobitaka. Da bi to bilo moguće neophodno je negirati sve vrednosti funkcije procene dobijene od svih pozicija naslednika tekuće pozicije. Ovo je izvodljivo zbog Zero-sum svojstva igre.

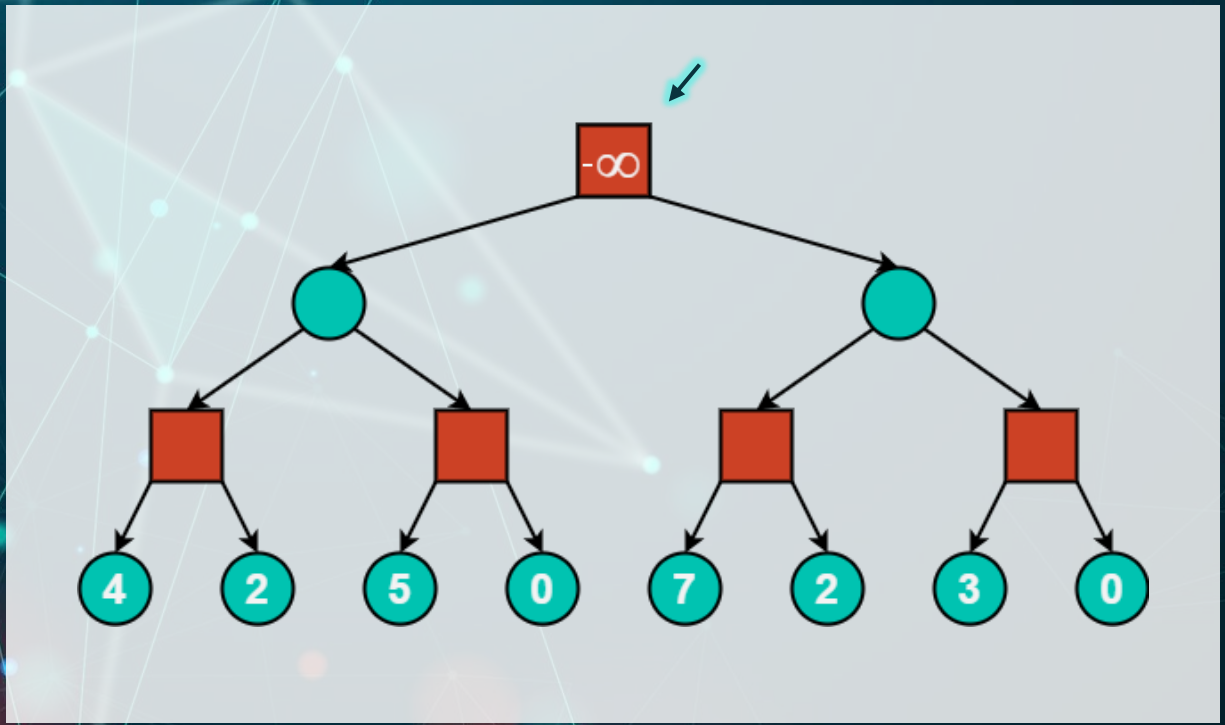
$$\max(x, y) = -\min(-x, -y)$$

NEGAMAX ALGORITAM

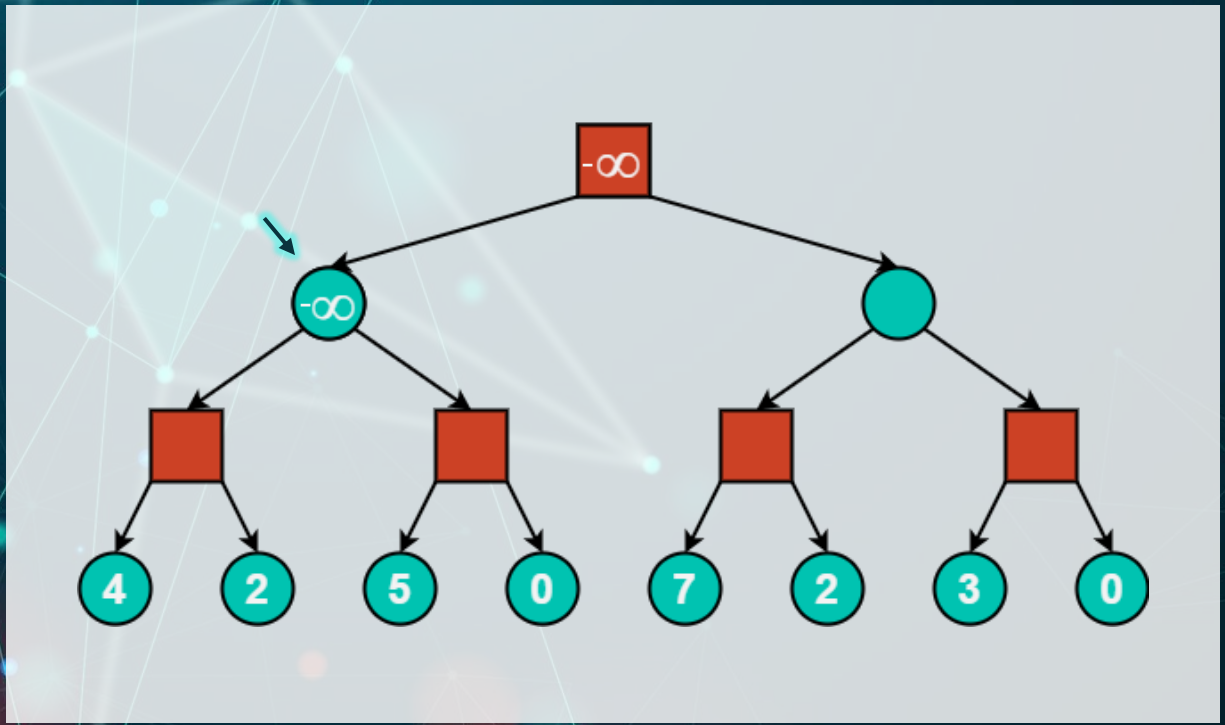
```
def negamax(node, player):
    if is_terminal_node(node):
        return node_evaluation(node) * (-1 if player == Player.MIN else 1)

    if player == Player.MAX:
    score = -math.inf
    for succ in node.successors():
        score = max(score, -negamax(succ, switch(player)))
    return score
    else:
        score = +math.inf
        for succ in node.successors():
            score = min(score, minimax(succ, Player.MAX))
        return val
```

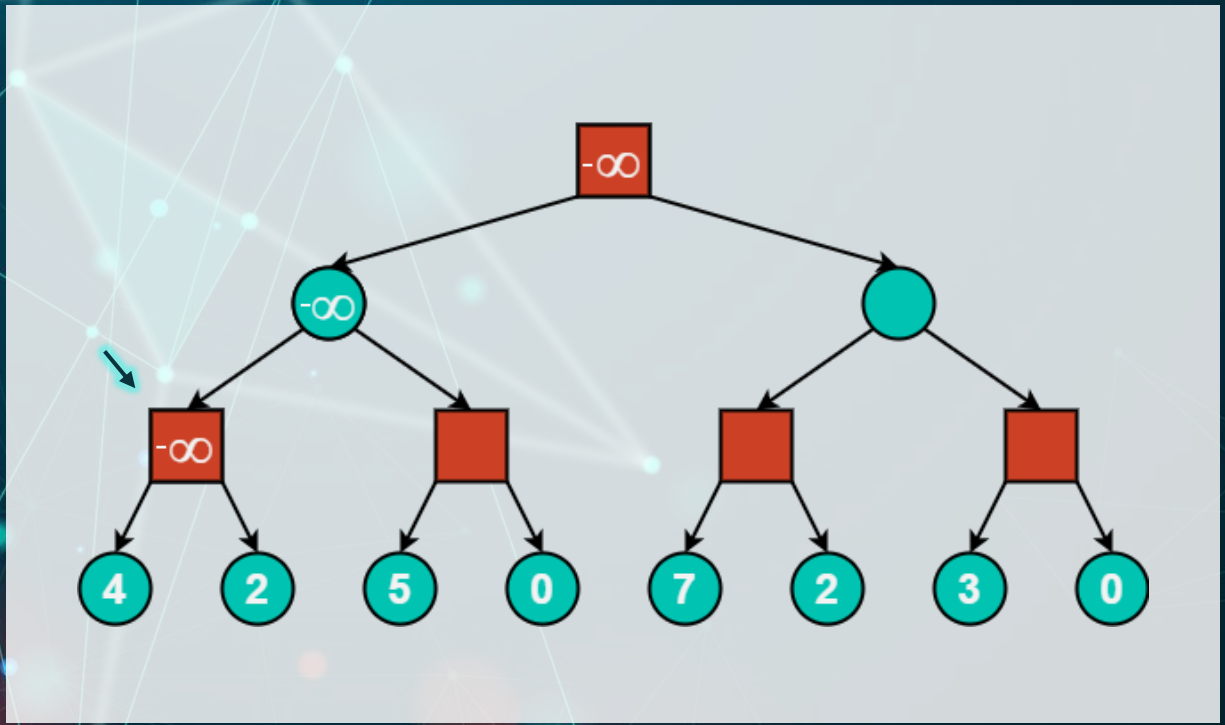
Zadatak 4 – Rešenje



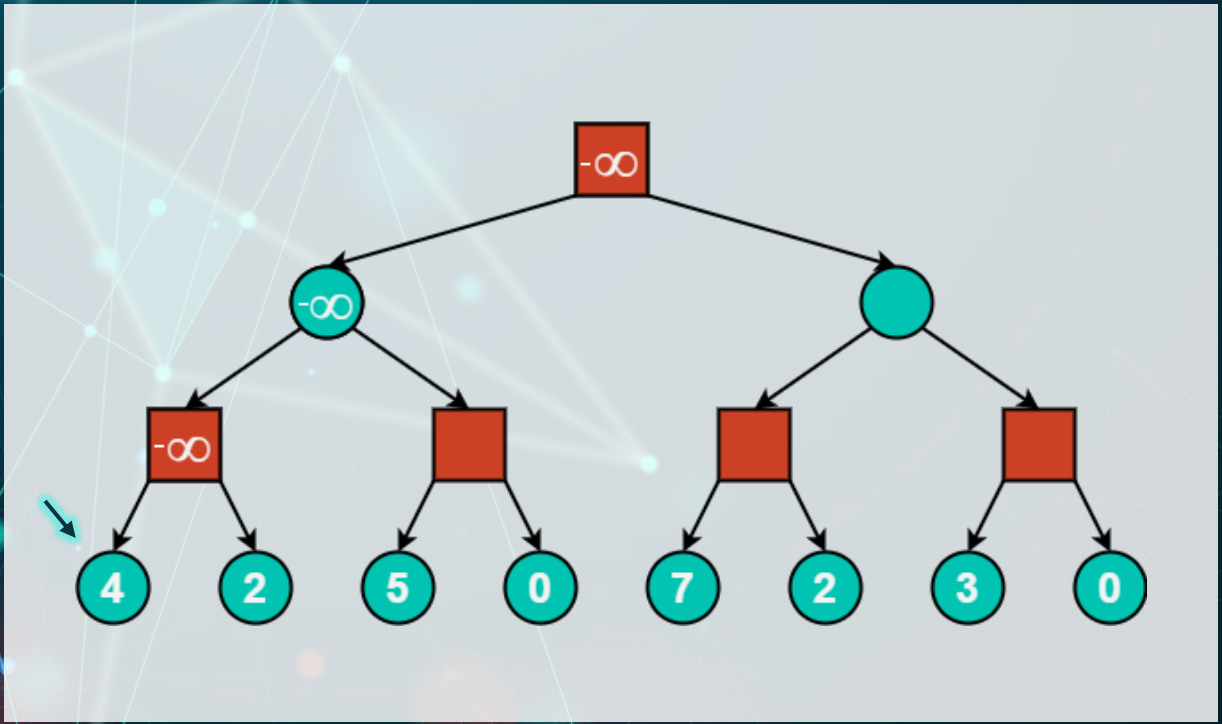
Zadatak 4 – Rešenje



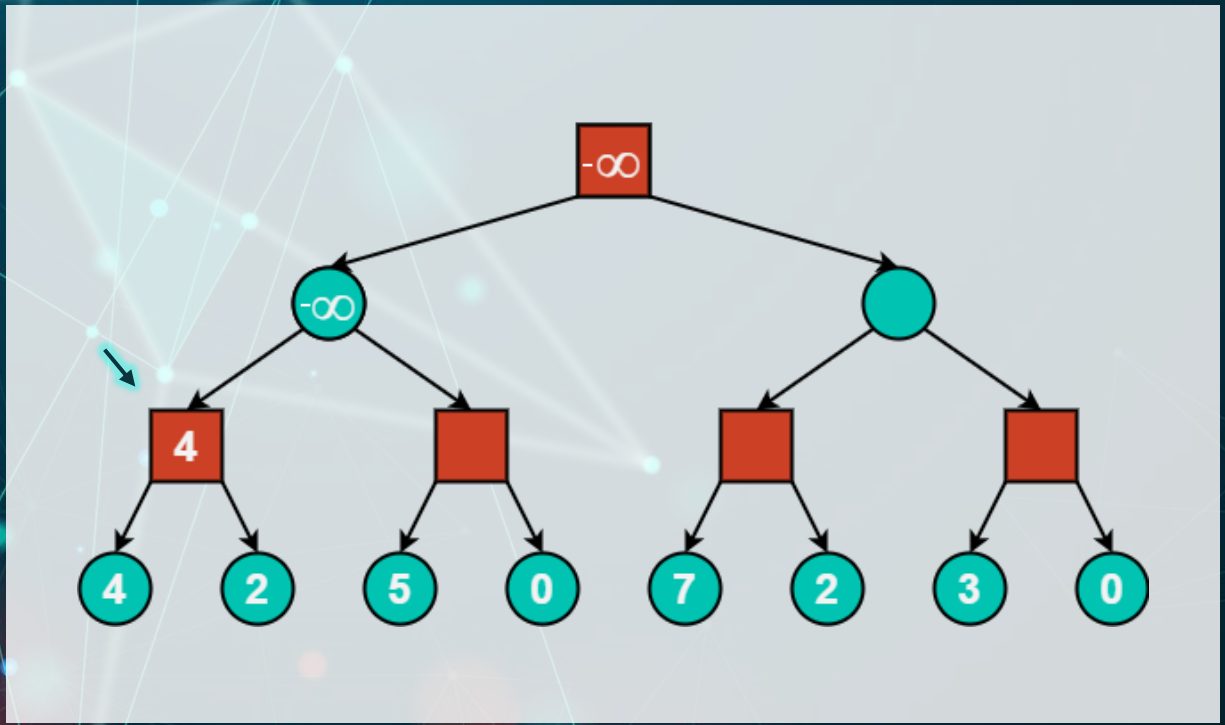
Zadatak 4 – Rešenje



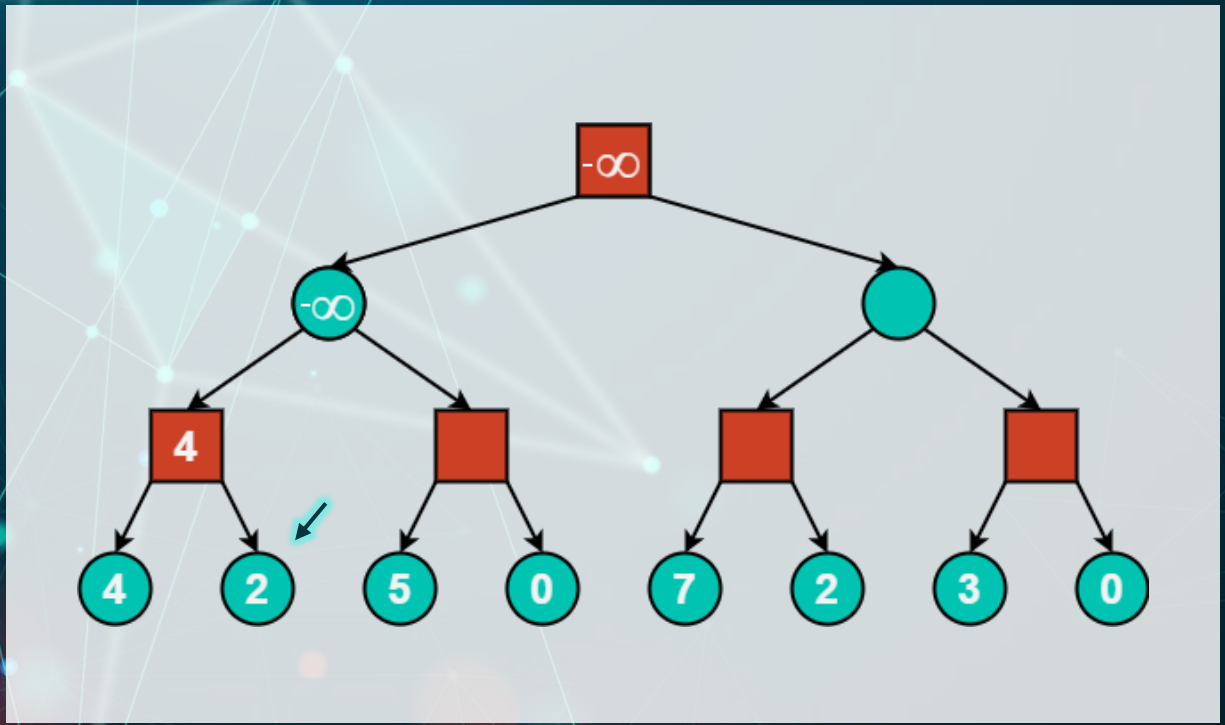
Zadatak 4 – Rešenje



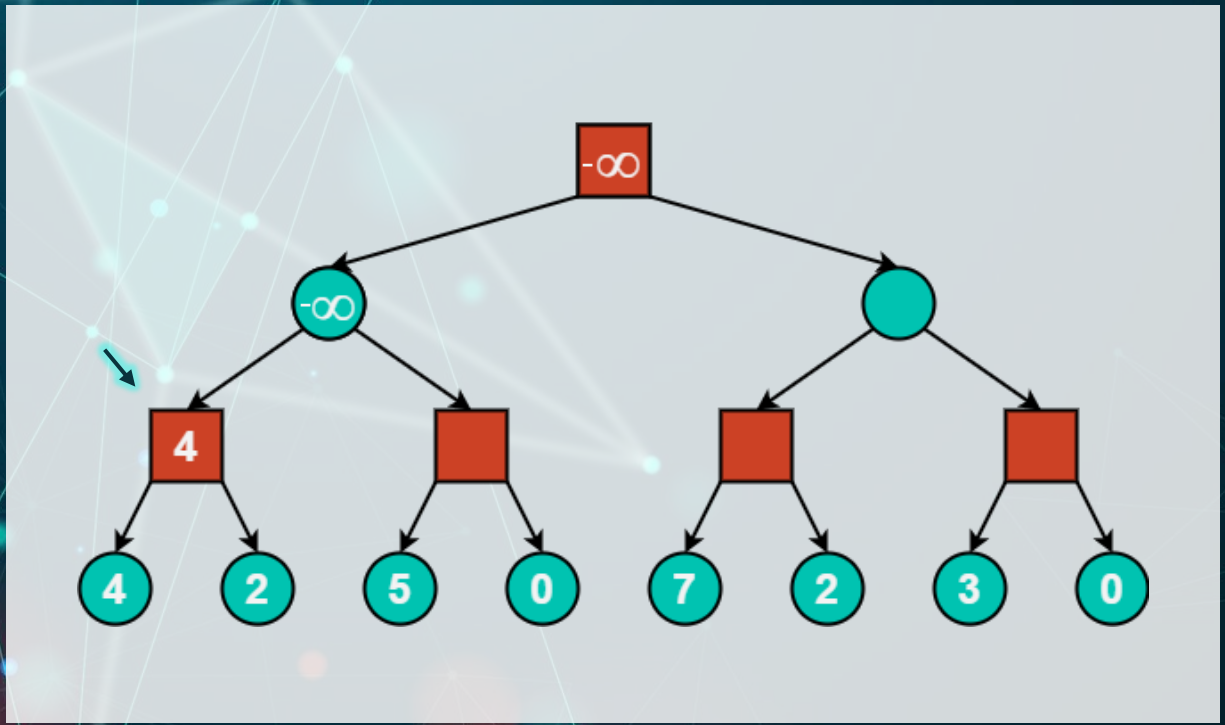
Zadatak 4 – Rešenje



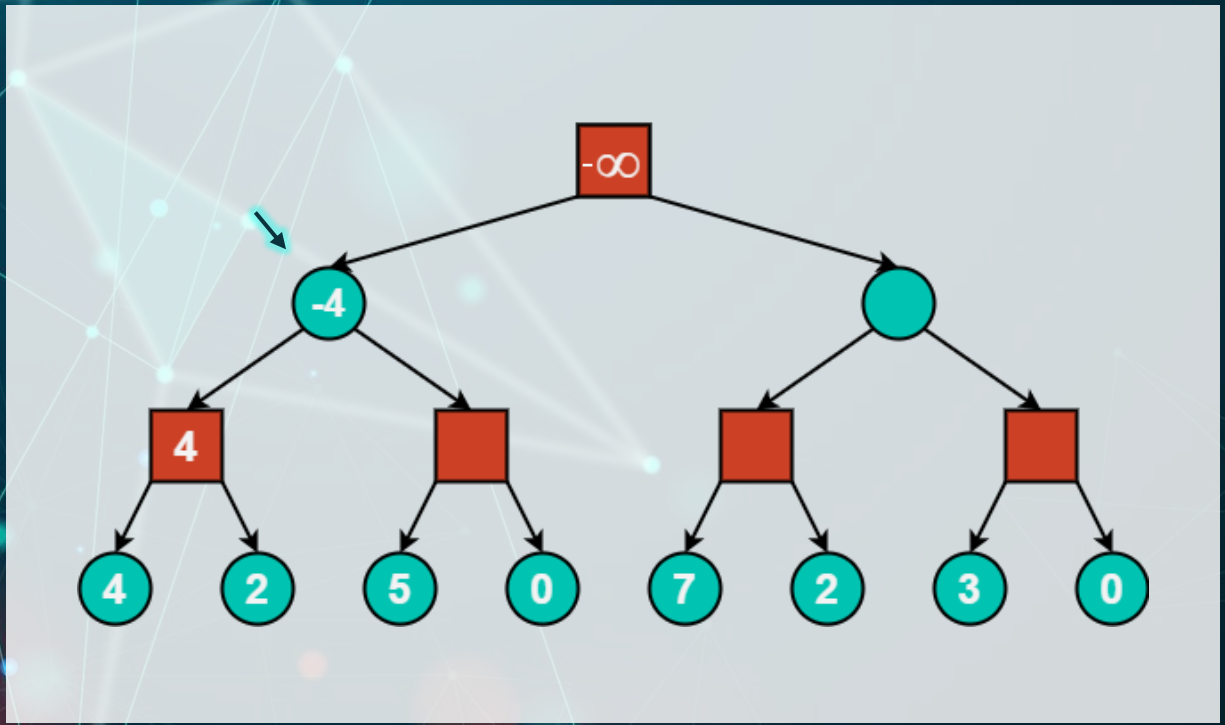
Zadatak 4 – Rešenje



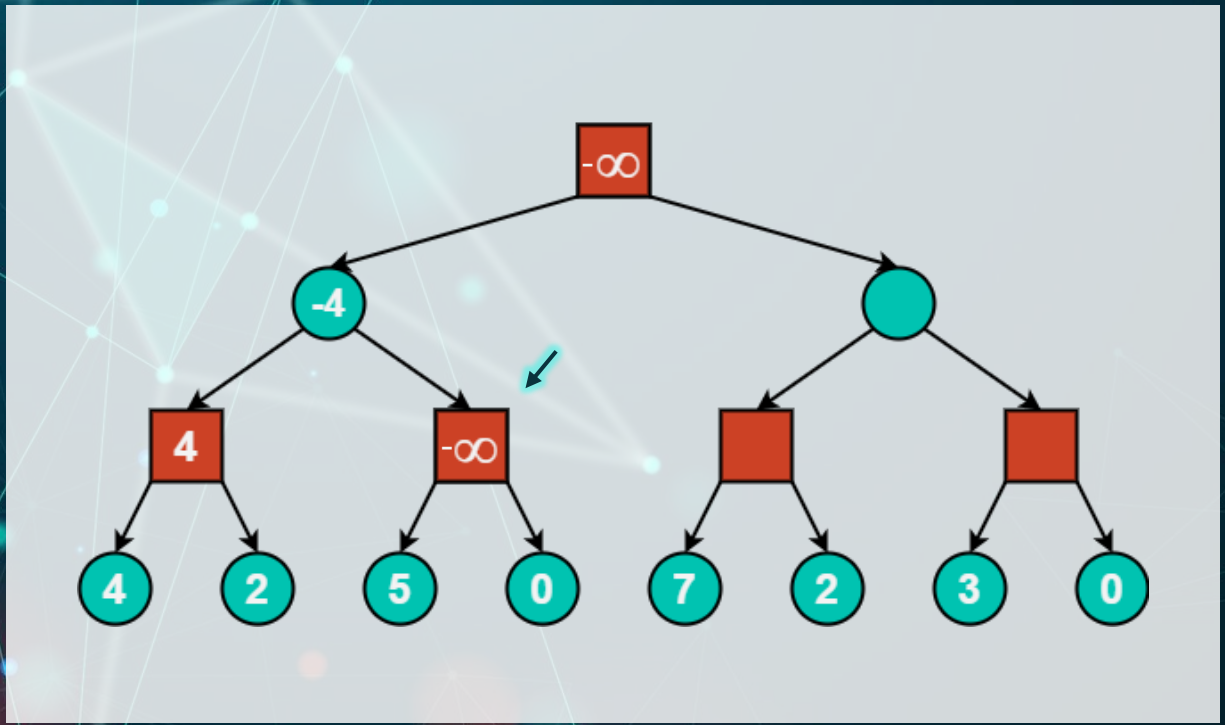
Zadatak 4 – Rešenje



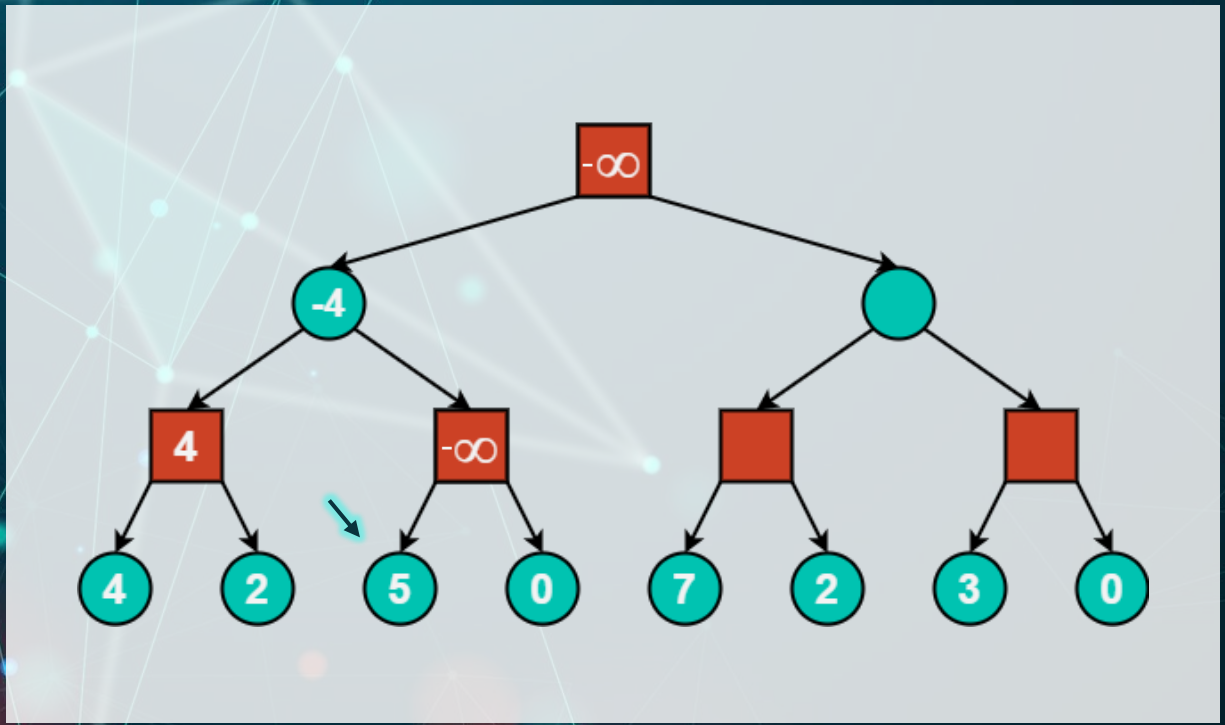
Zadatak 4 – Rešenje



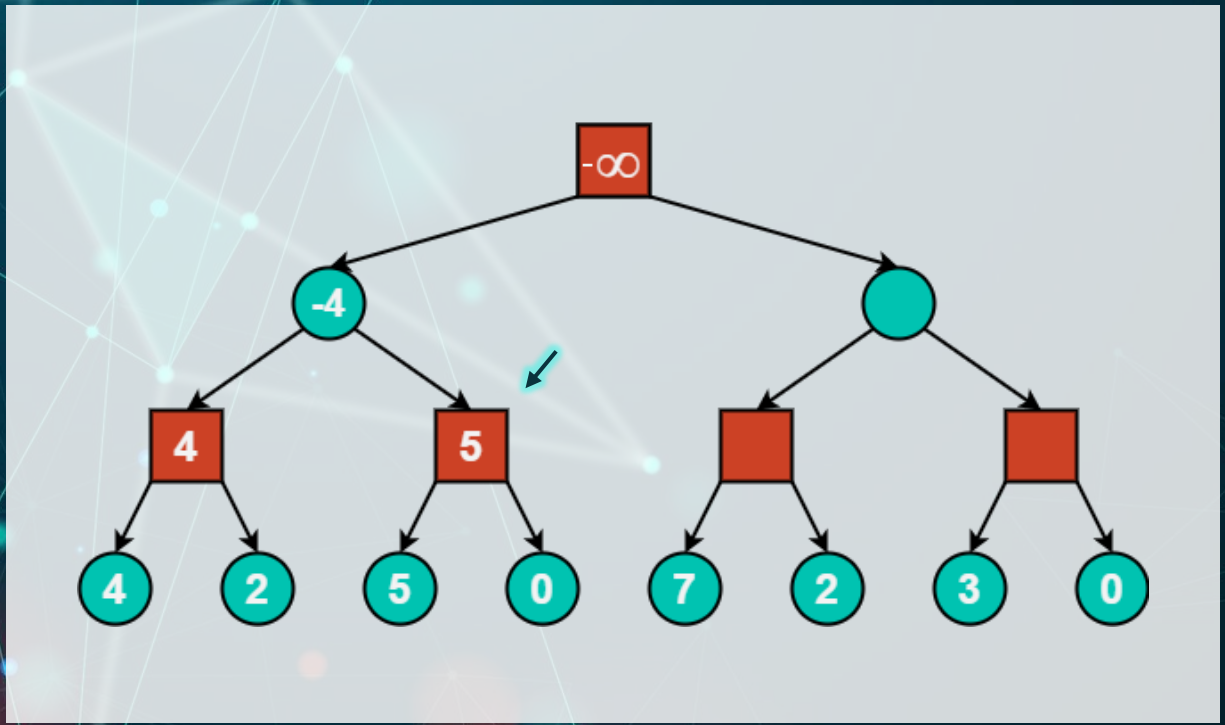
Zadatak 4 – Rešenje



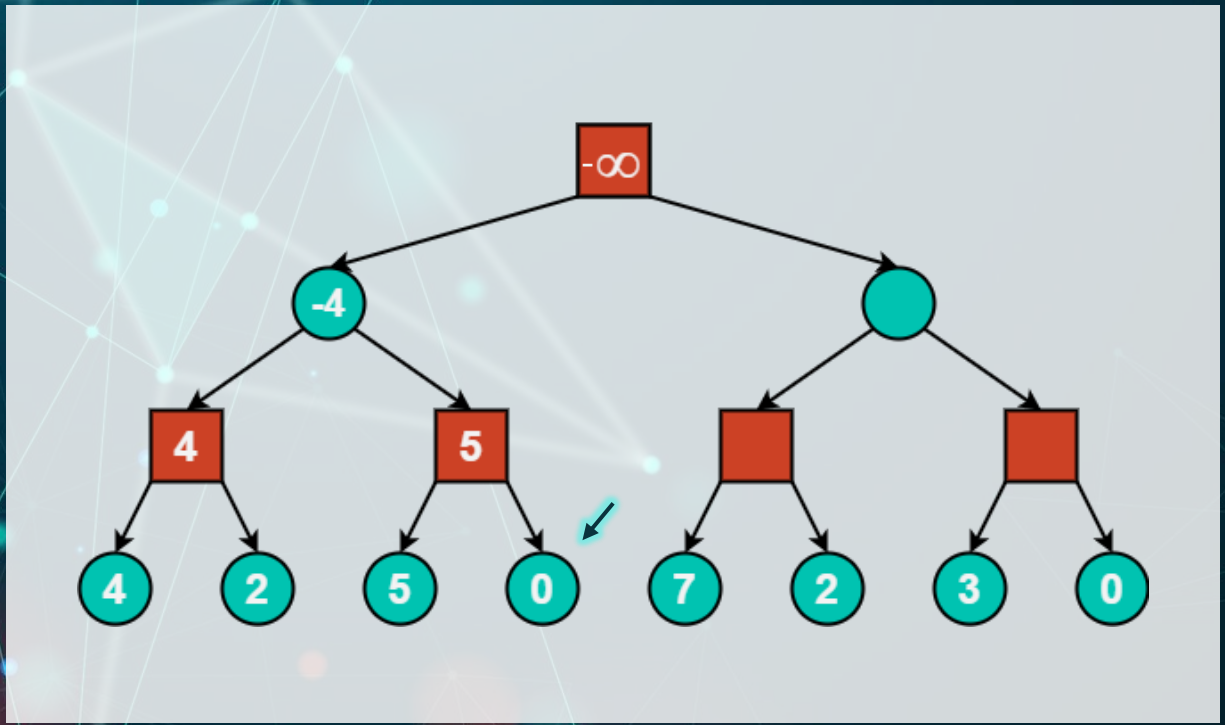
Zadatak 4 – Rešenje



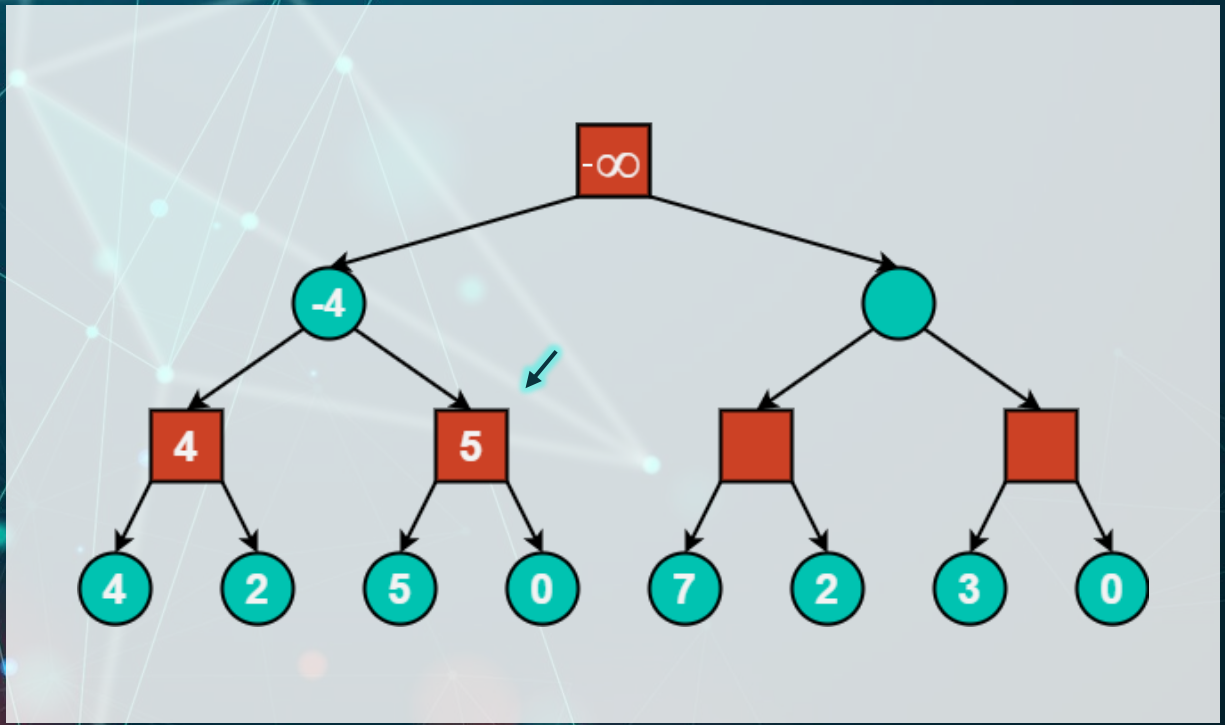
Zadatak 4 – Rešenje



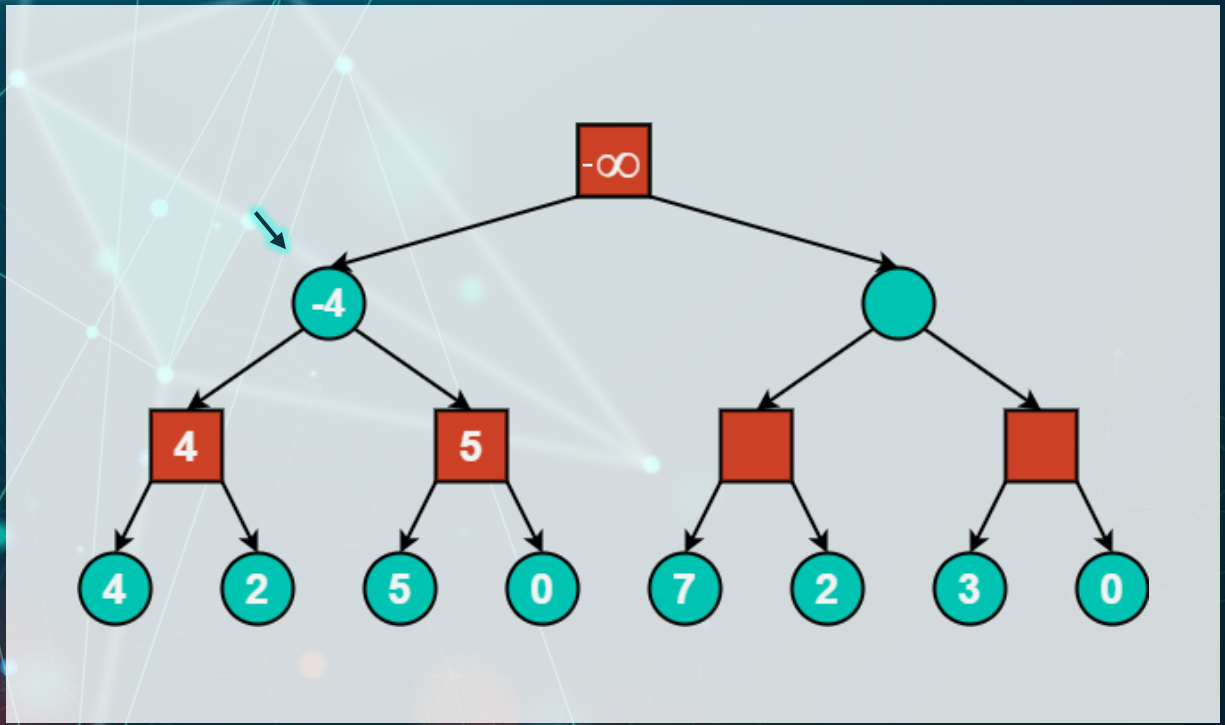
Zadatak 4 – Rešenje



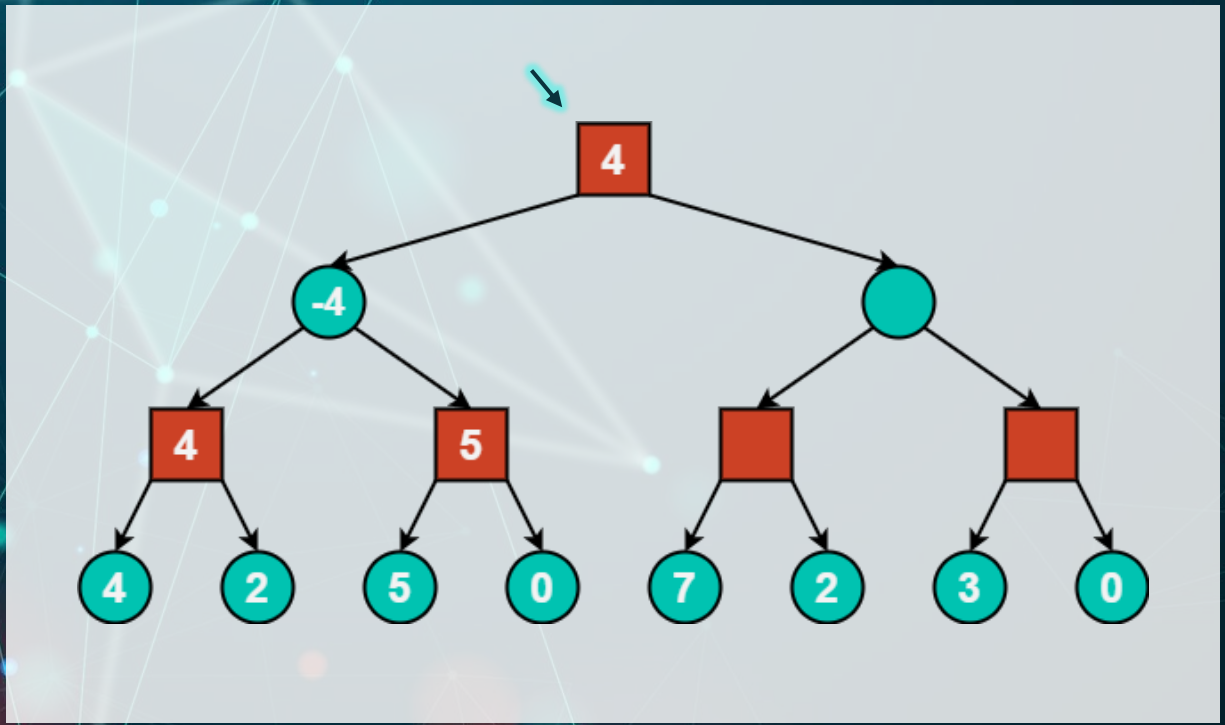
Zadatak 4 – Rešenje



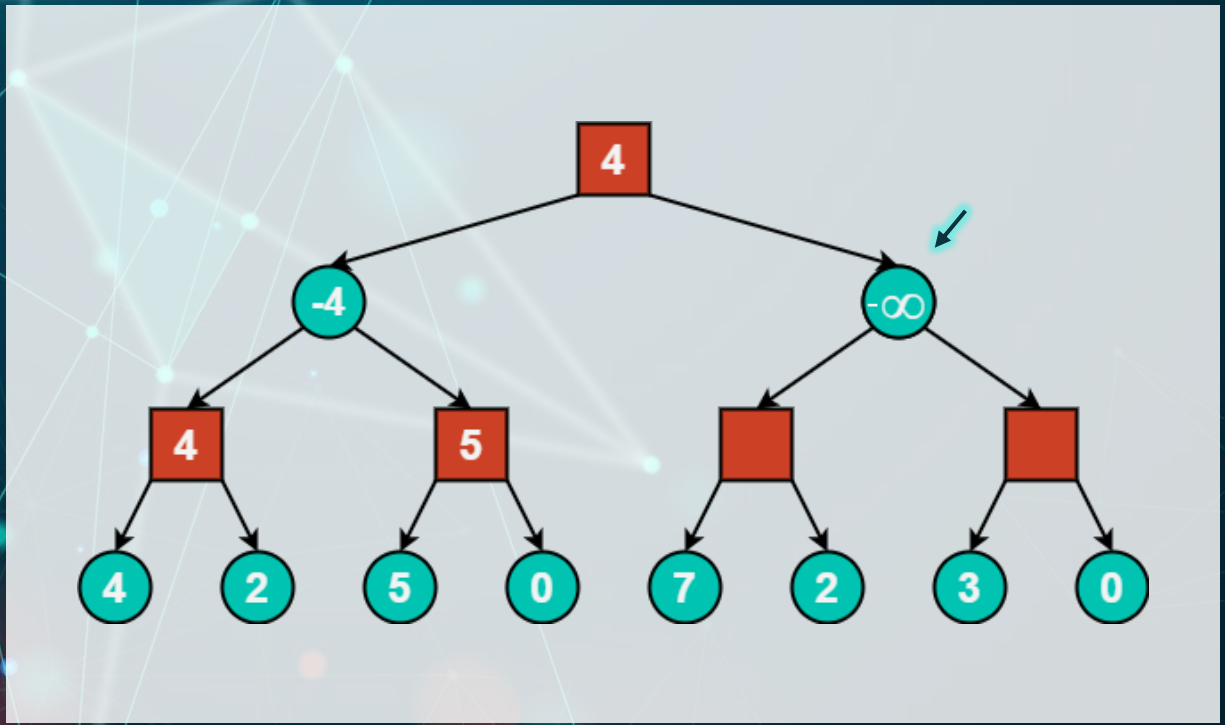
Zadatak 4 – Rešenje



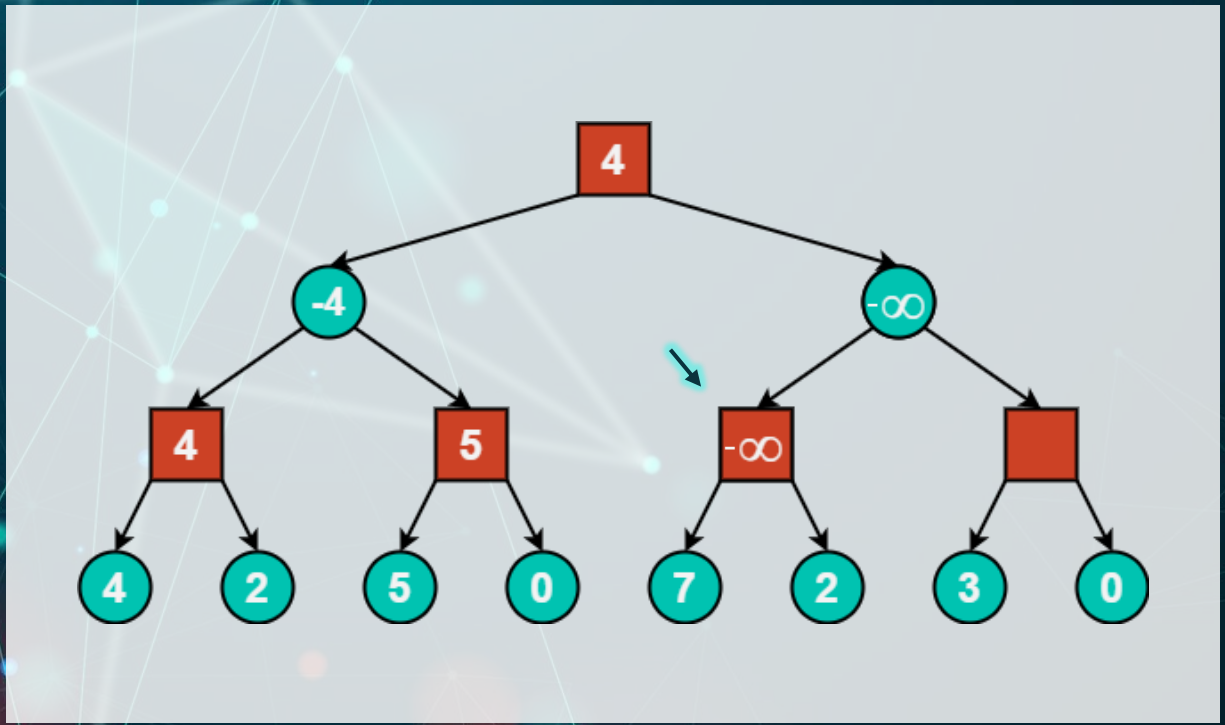
Zadatak 4 – Rešenje



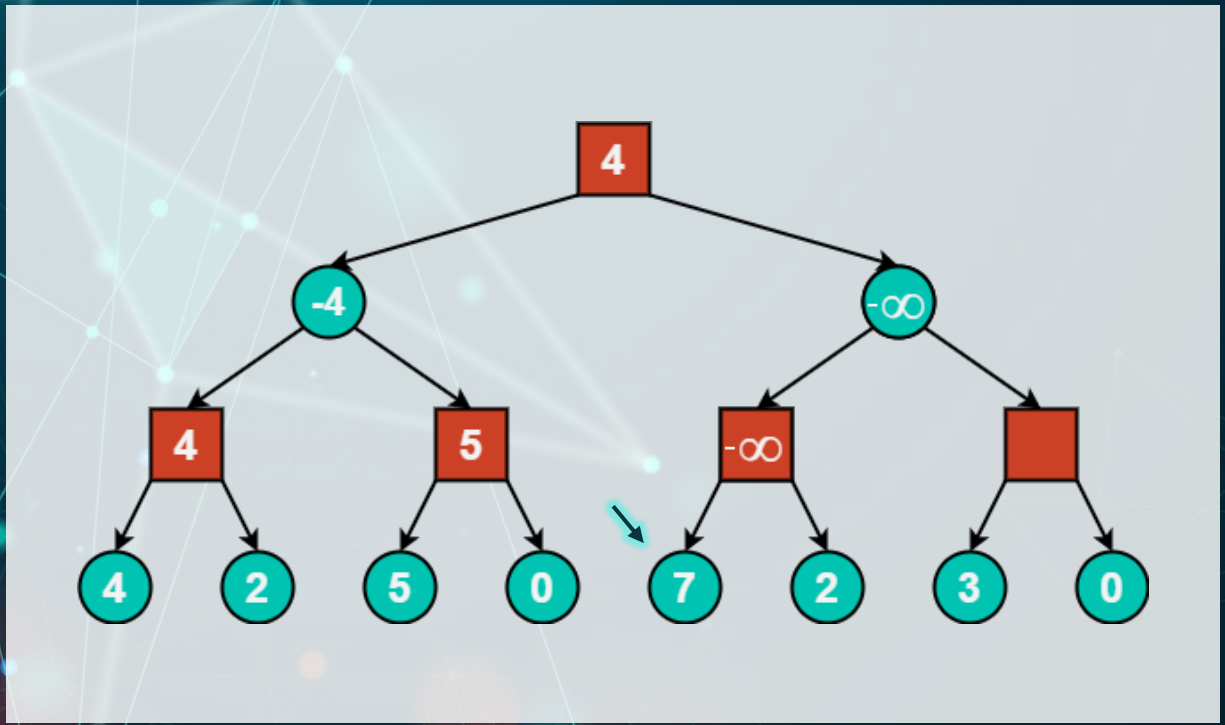
Zadatak 4 – Rešenje



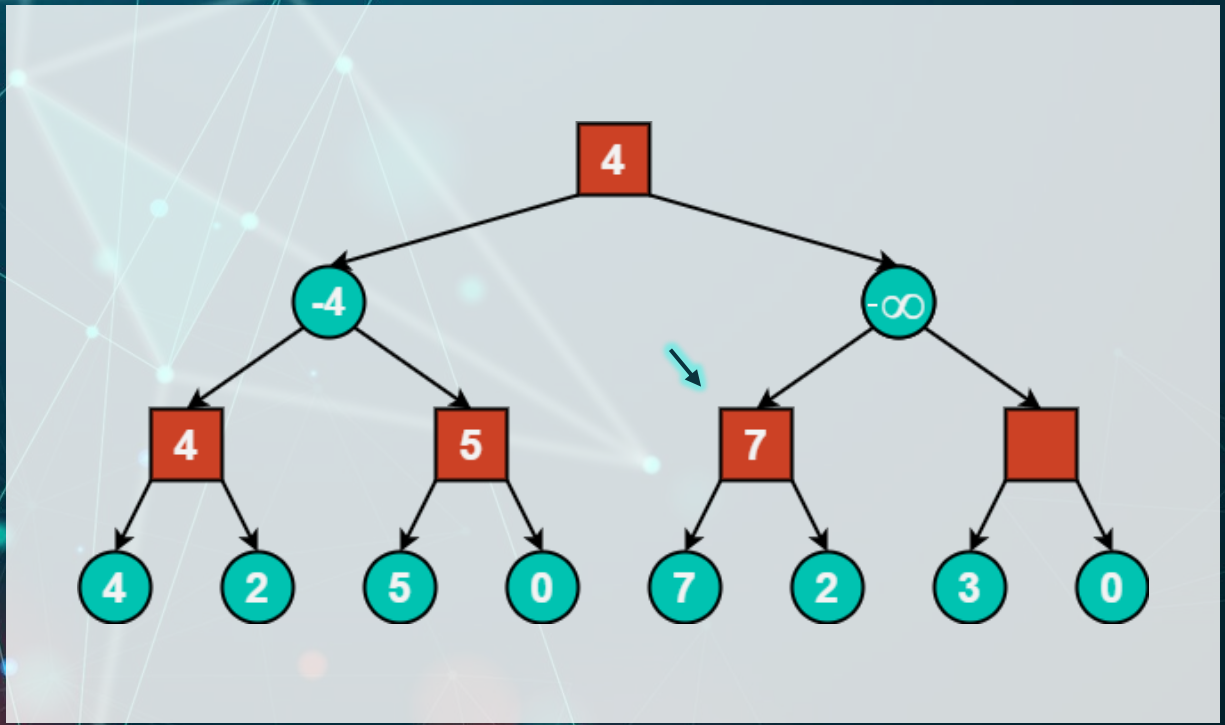
Zadatak 4 – Rešenje



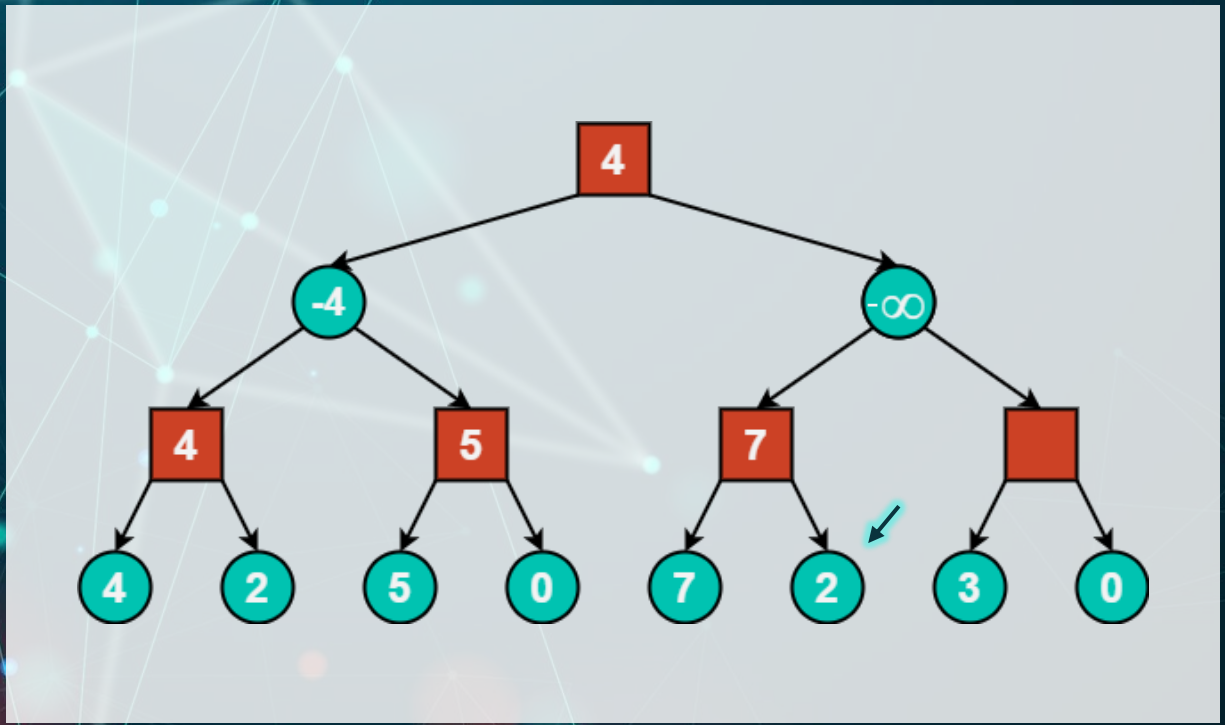
Zadatak 4 – Rešenje



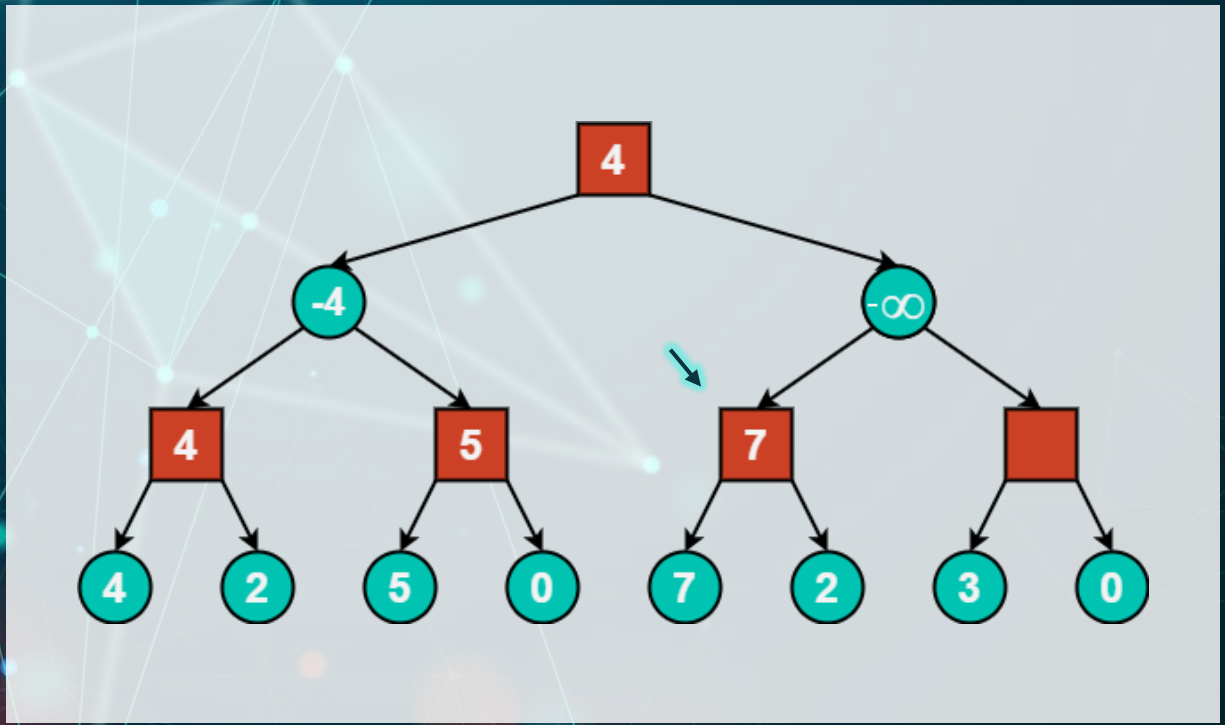
Zadatak 4 – Rešenje



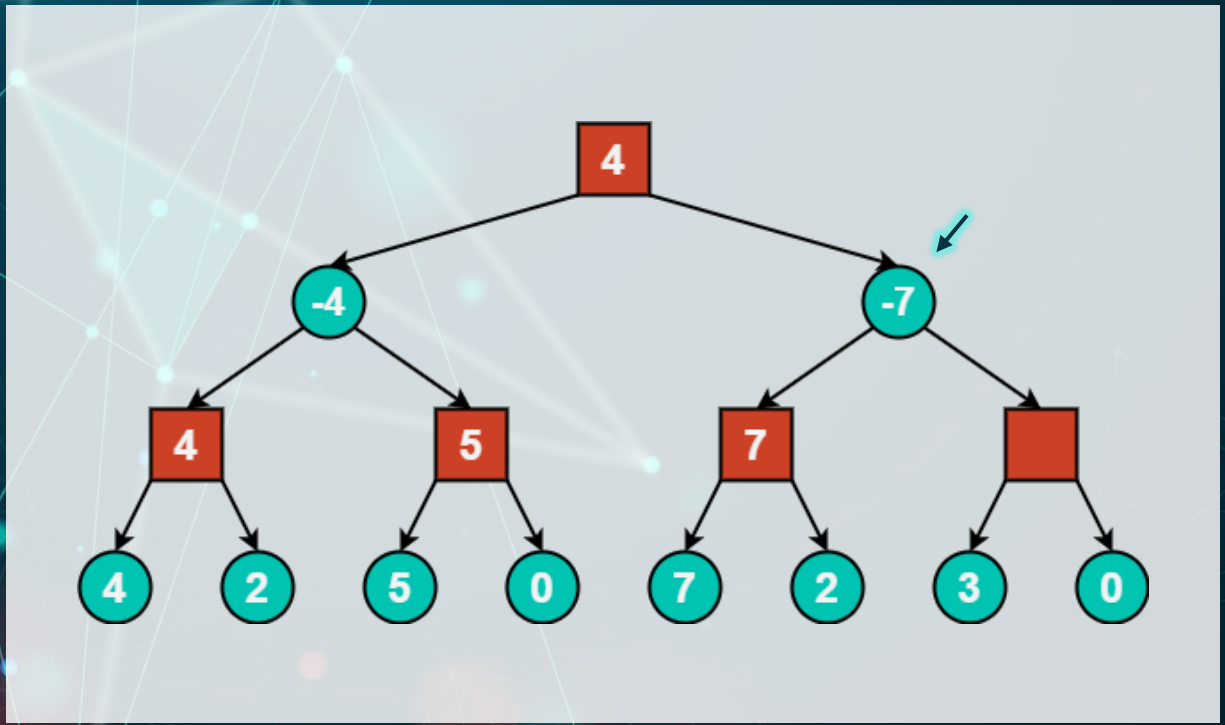
Zadatak 4 – Rešenje



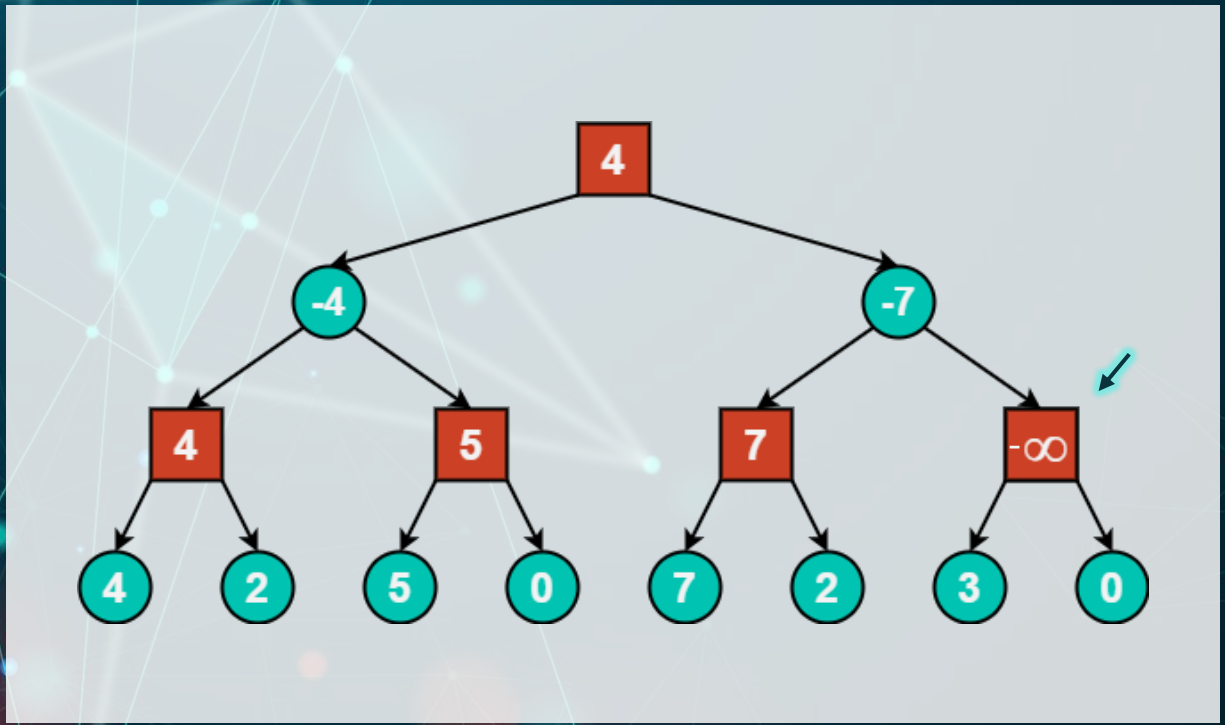
Zadatak 4 – Rešenje



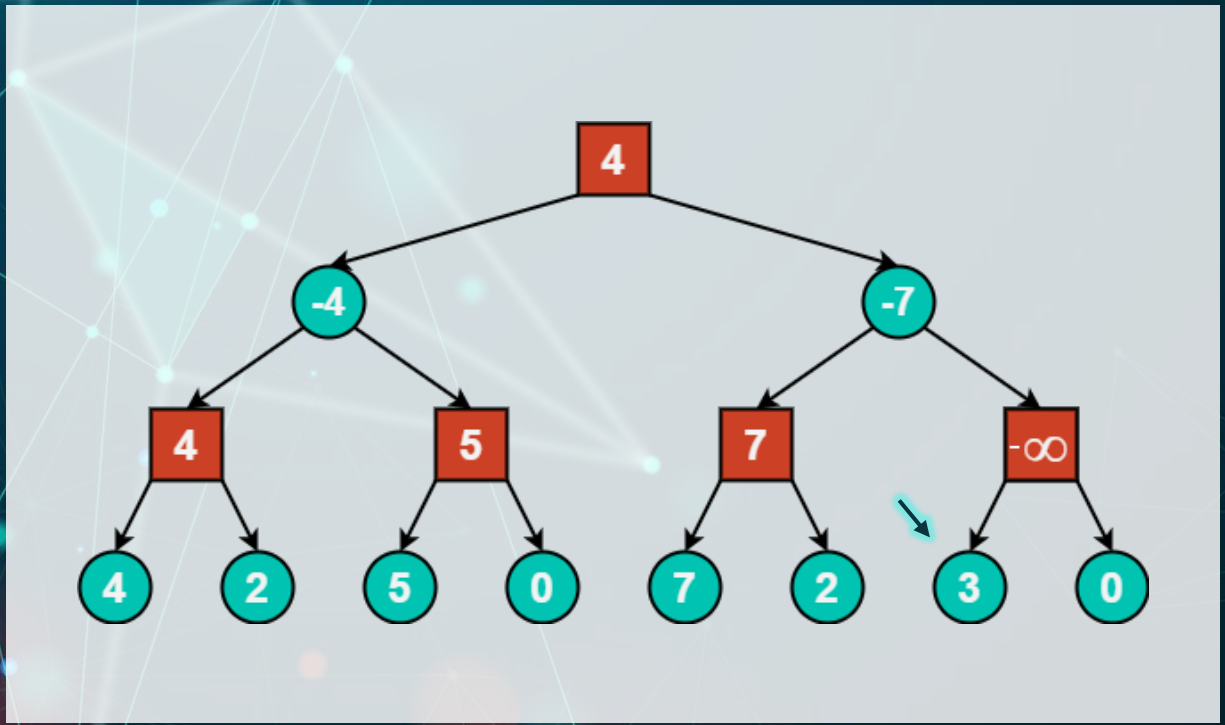
Zadatak 4 – Rešenje



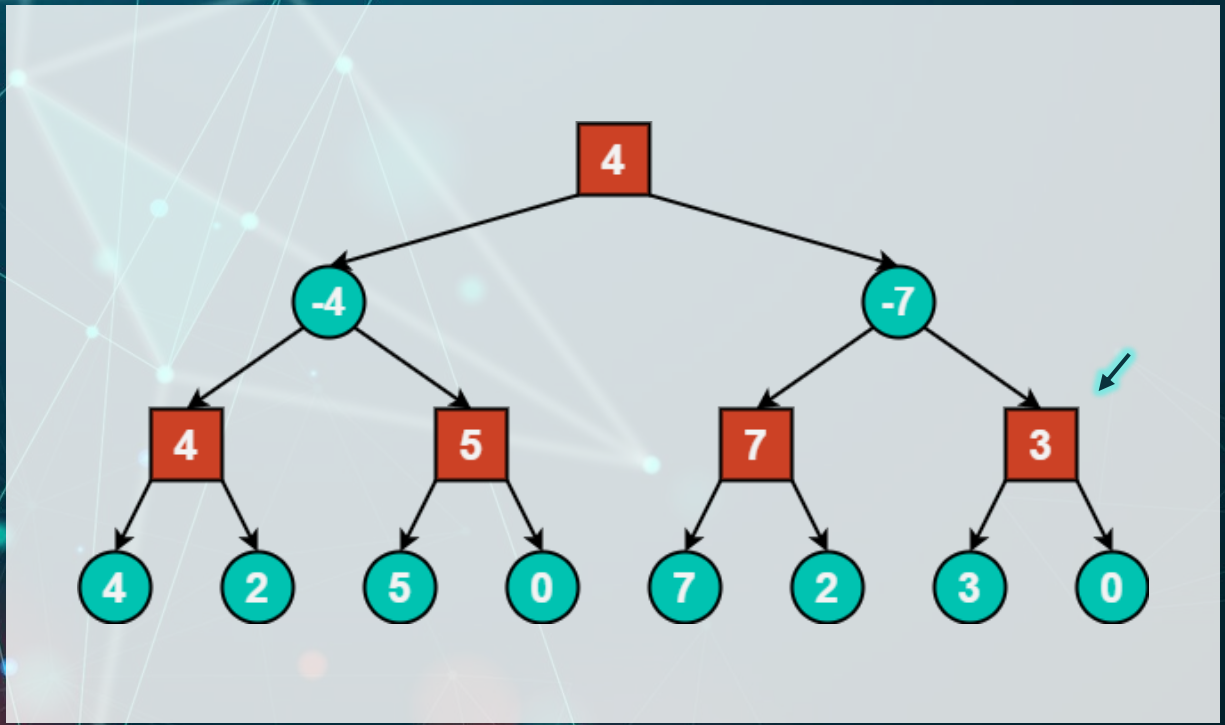
Zadatak 4 – Rešenje



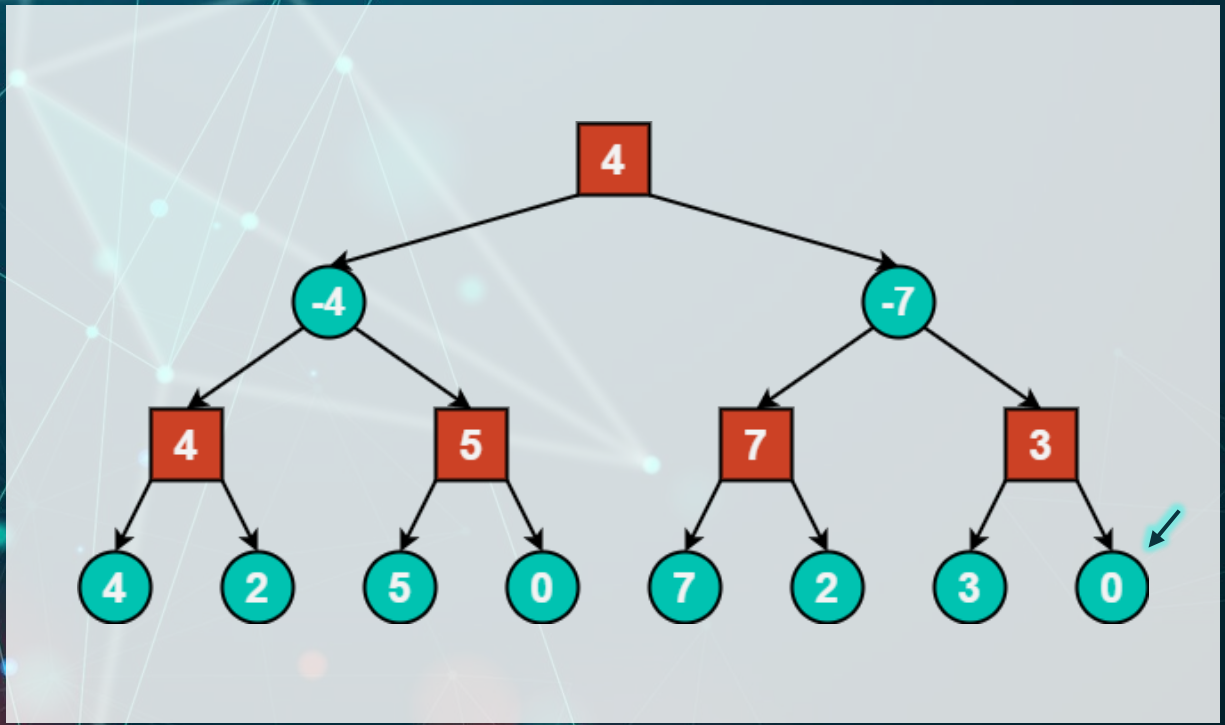
Zadatak 4 – Rešenje



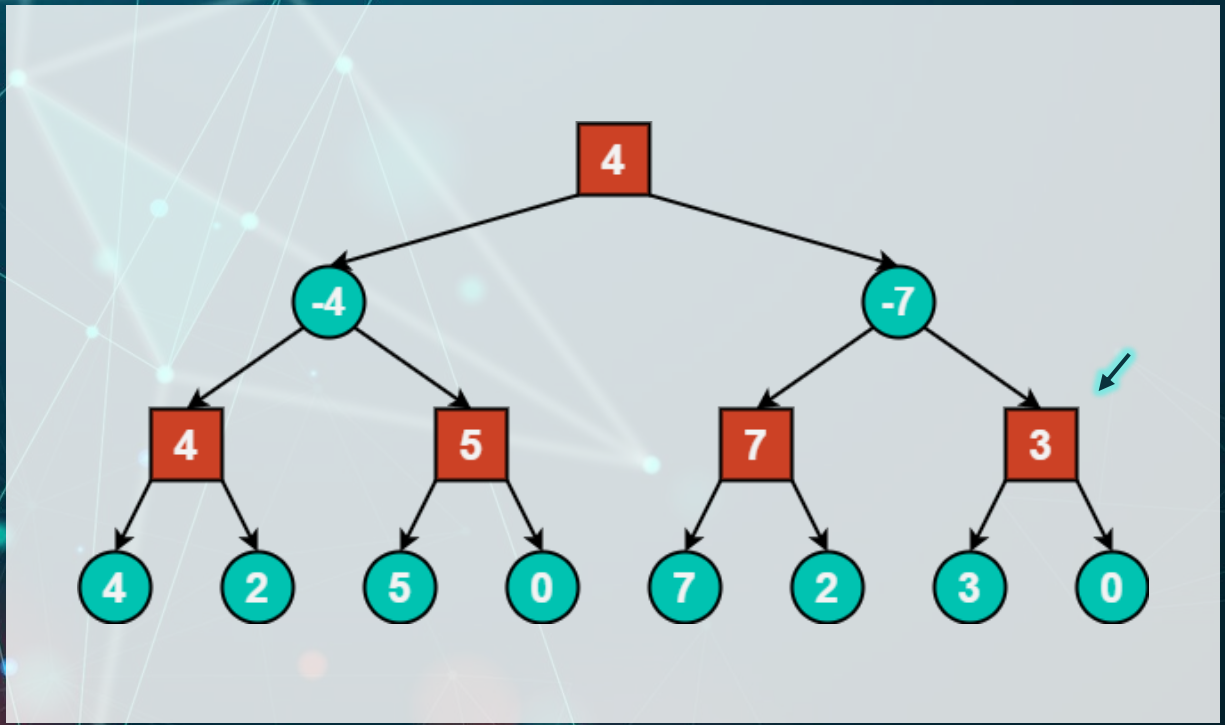
Zadatak 4 – Rešenje



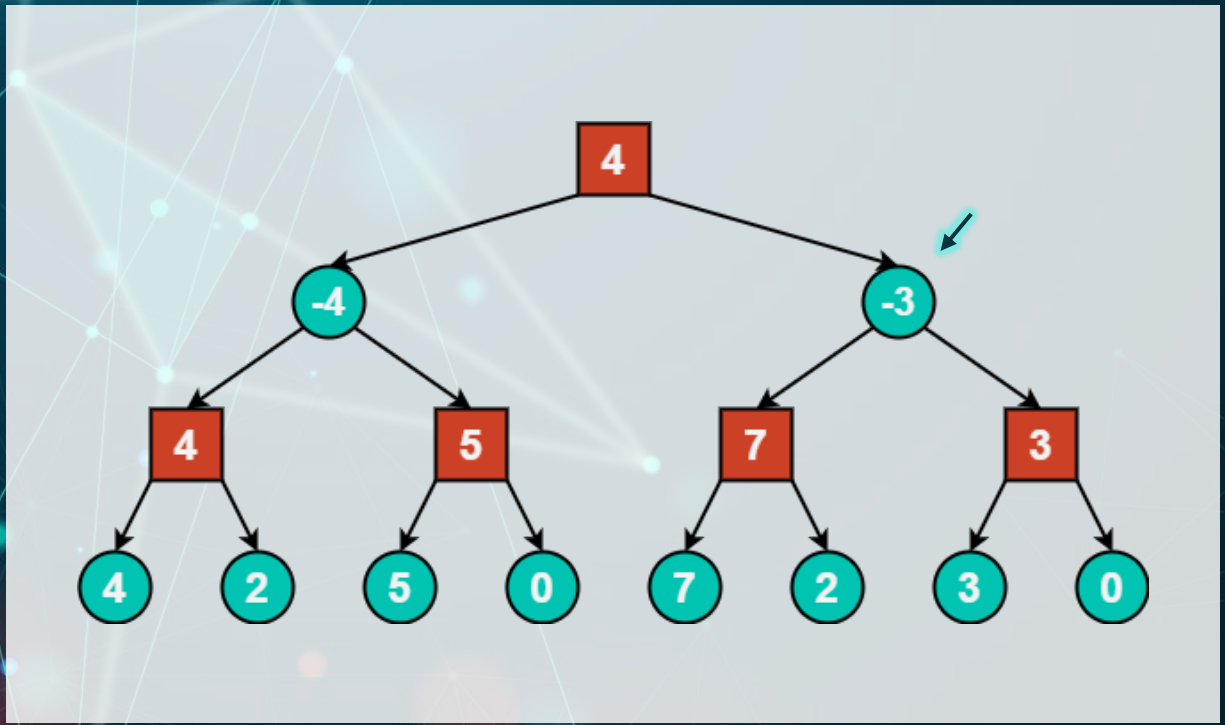
Zadatak 4 – Rešenje



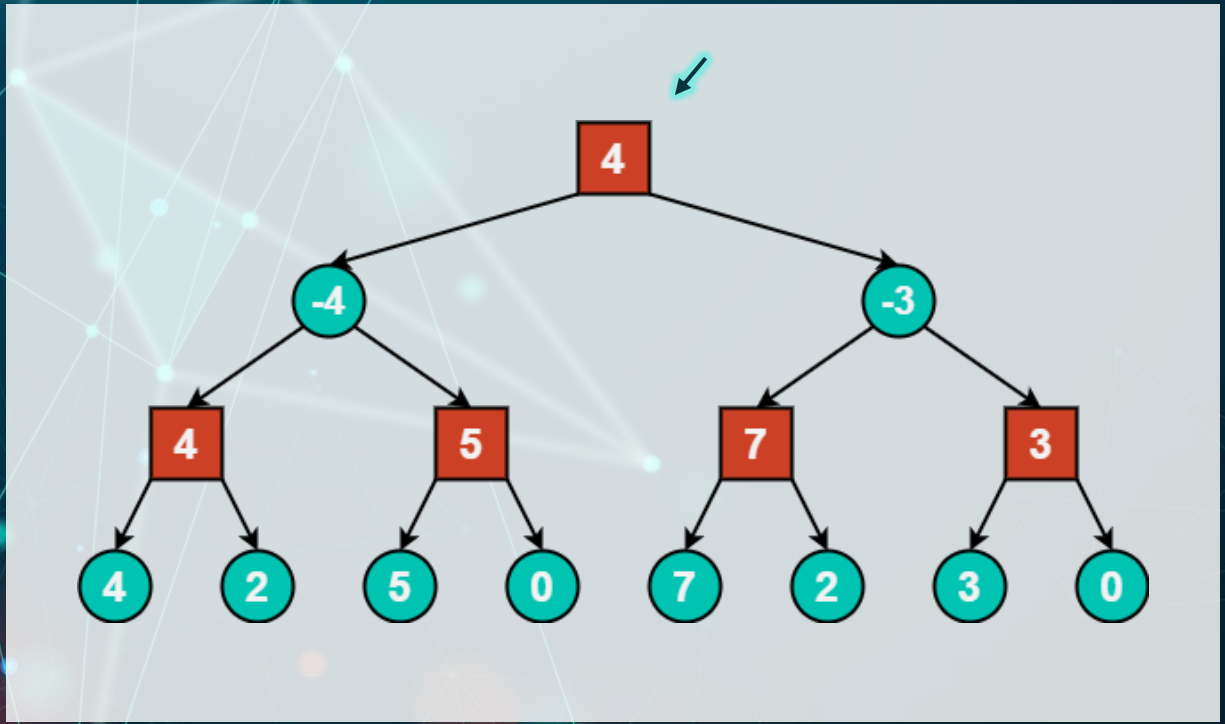
Zadatak 4 – Rešenje



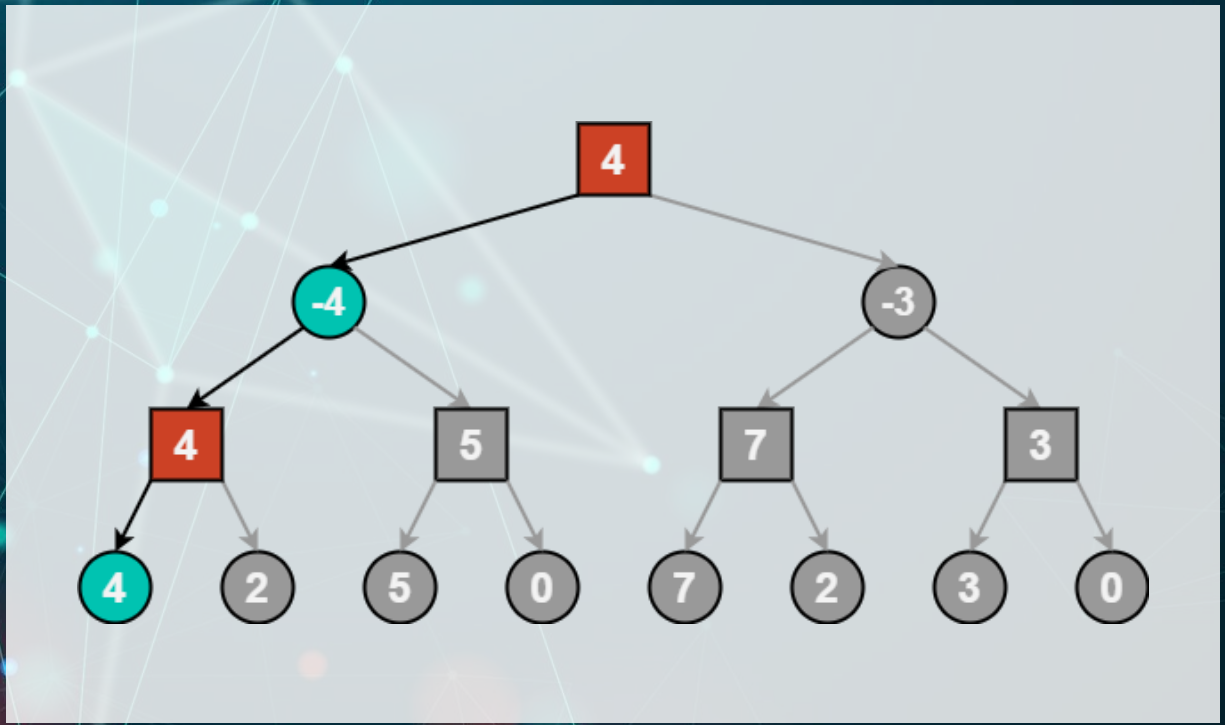
Zadatak 4 – Rešenje



Zadatak 4 – Rešenje



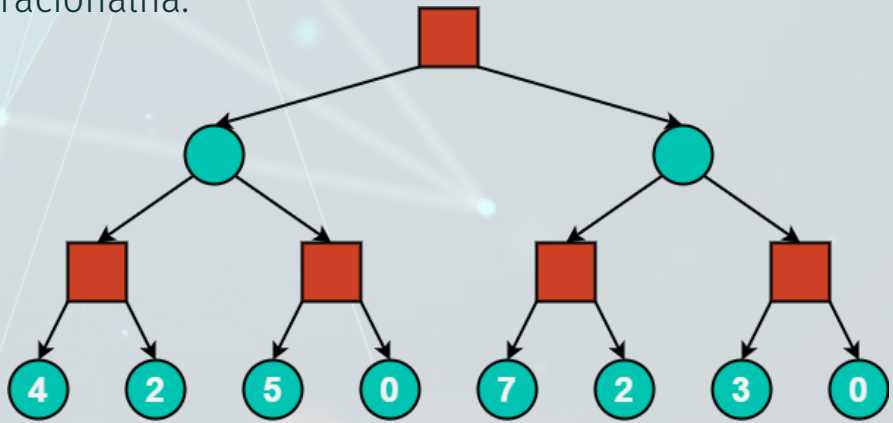
Zadatak 4 – Rešenje



Zadatak 5 – Negamax (alfa beta)

★★★★☆

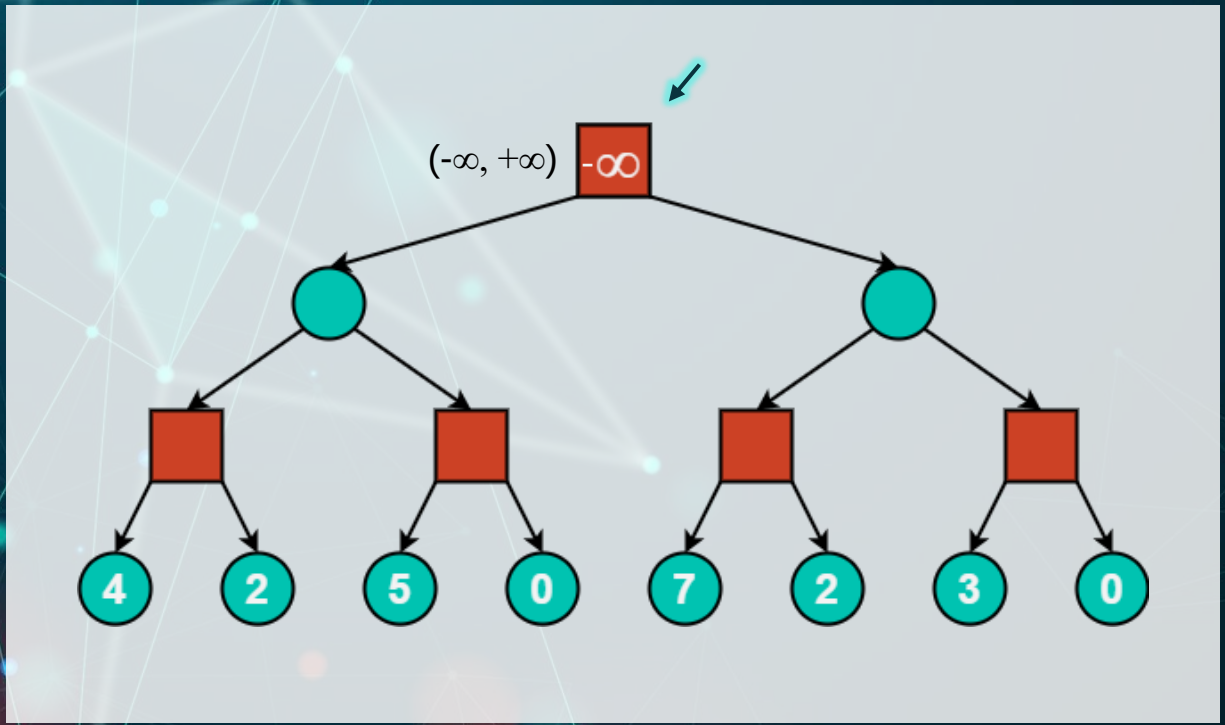
Dato je kompletno stablo igre sa naznačenim dobicima za prvog igrača. Primenom **negamax** algoritma sa alfa-beta odsecanjem odrediti funkciju procene za svaki čvor, konačan ishod igre i sekvencu poteza koja će biti odigrana do kraja ukoliko su oba igrača racionalna.



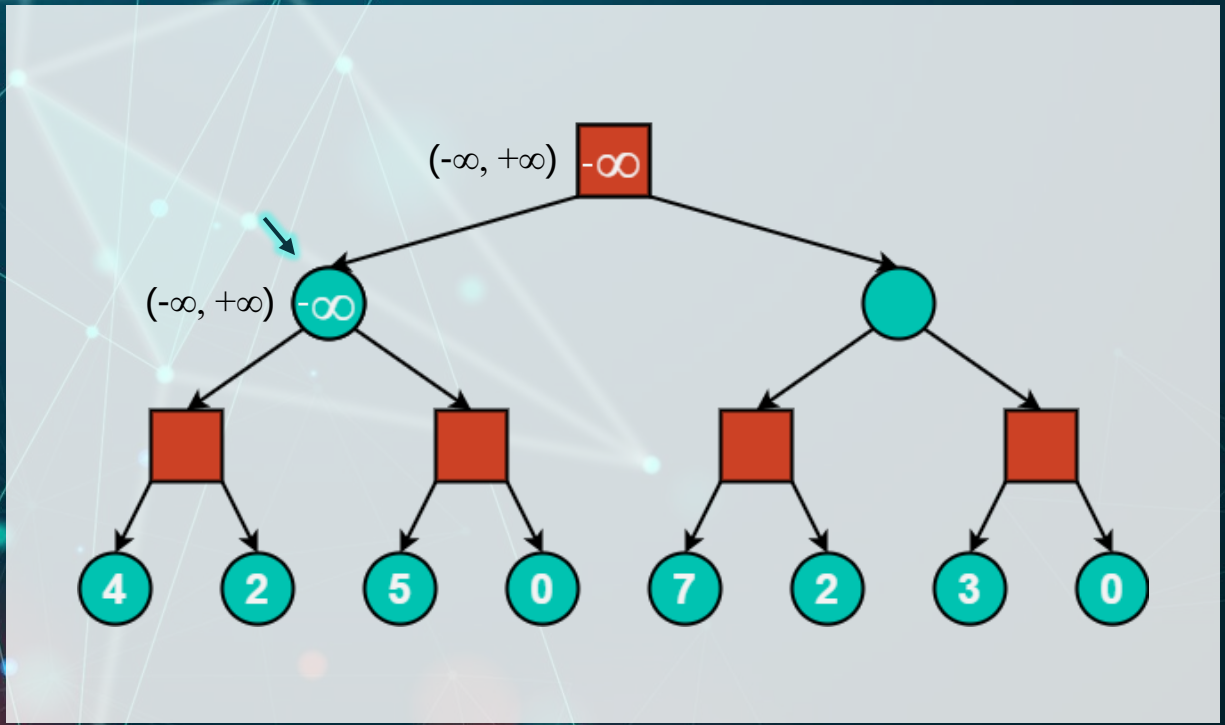
NEGAMAX ALPHA-BETA ALGORITAM

```
def negamax_alpha_beta(node, player, alpha, beta):
    if is_terminal_node(node):
        return node_evaluation(node) * (-1 if player == Player.MIN else 1)
    if player == Player.MAX:
    score = -math.inf
    for succ in node.successors():
        val = -negamax_alpha_beta(succ, switch(player), -beta, -alpha)
        score = max(score, val)
        alpha = max(alpha, score)
        if alpha >= beta : break
    return score
    else:
        score = +math.inf
        for succ in node.successors():
            score = min(score, minimax(succ, Player.MAX, alpha, beta))
            beta = min(beta, score)
            if alpha >= beta: break
        return score
```

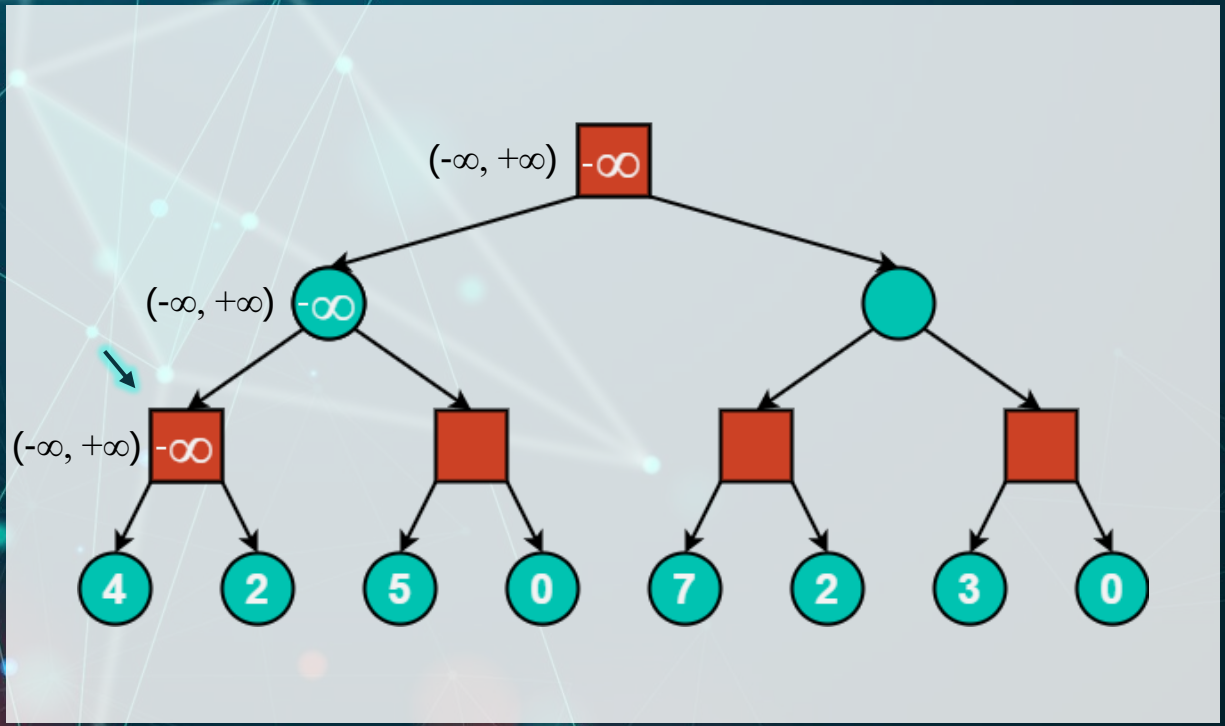
Zadatak 5- Rešenje



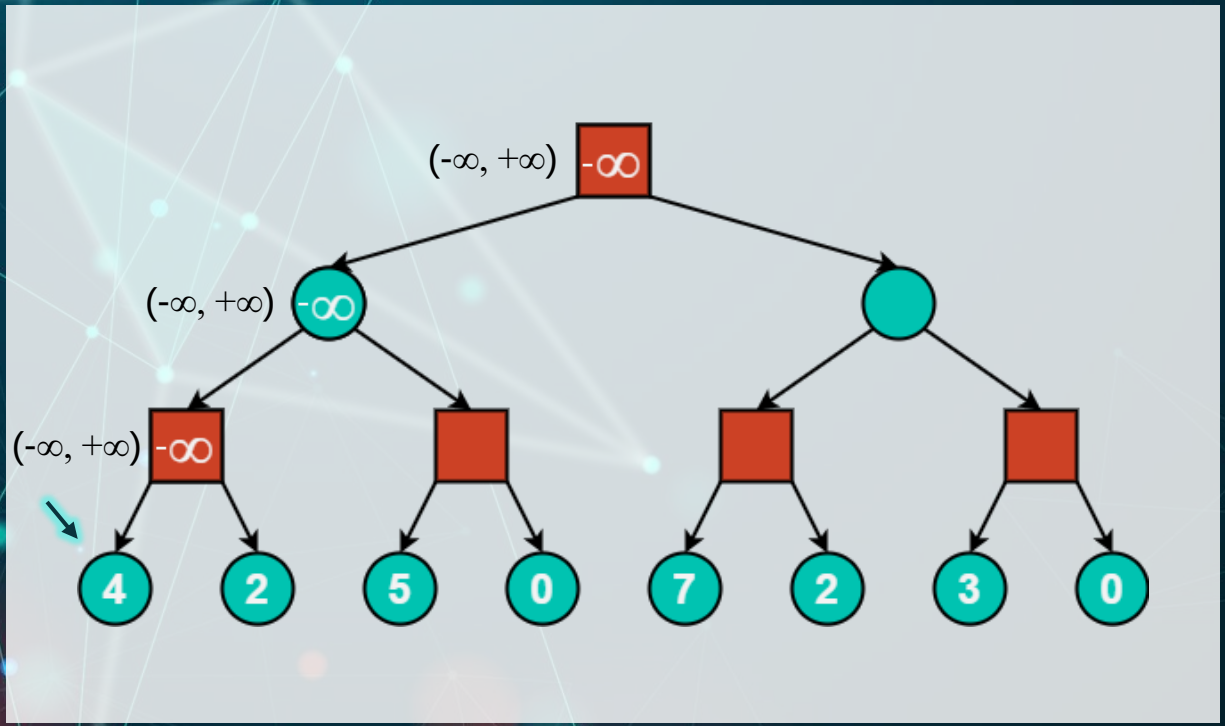
Zadatak 5 – Rešenje



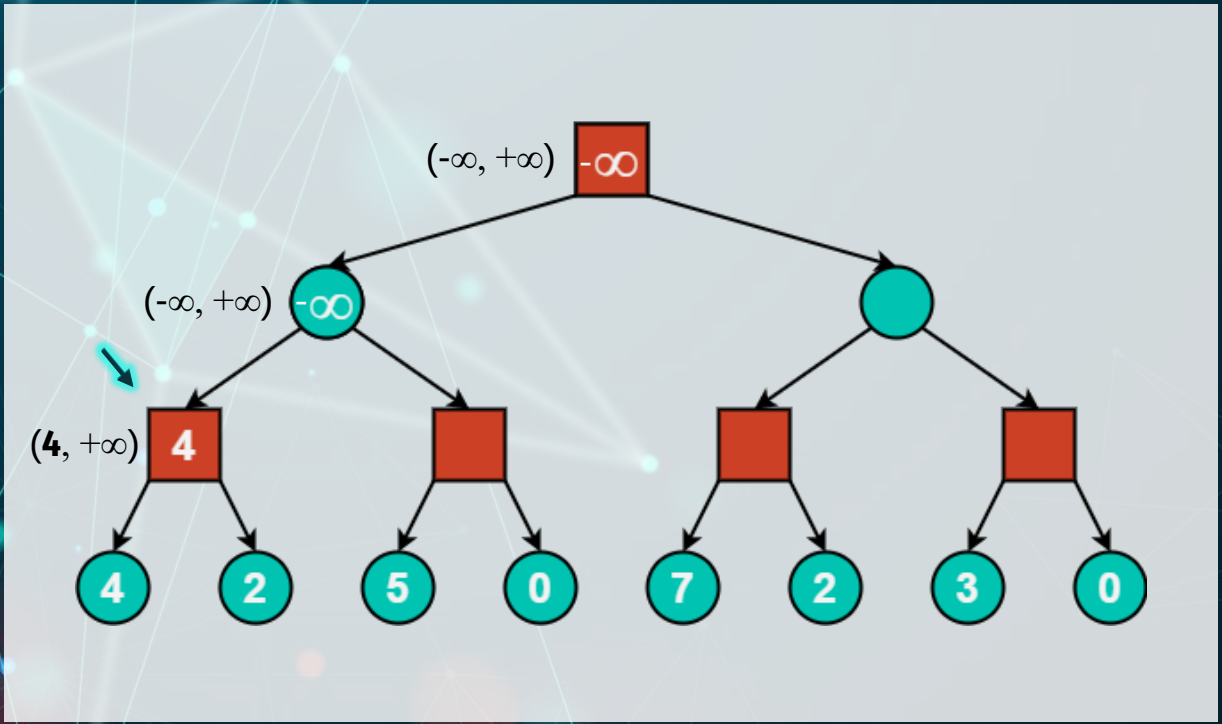
Zadatak 5 – Rešenje



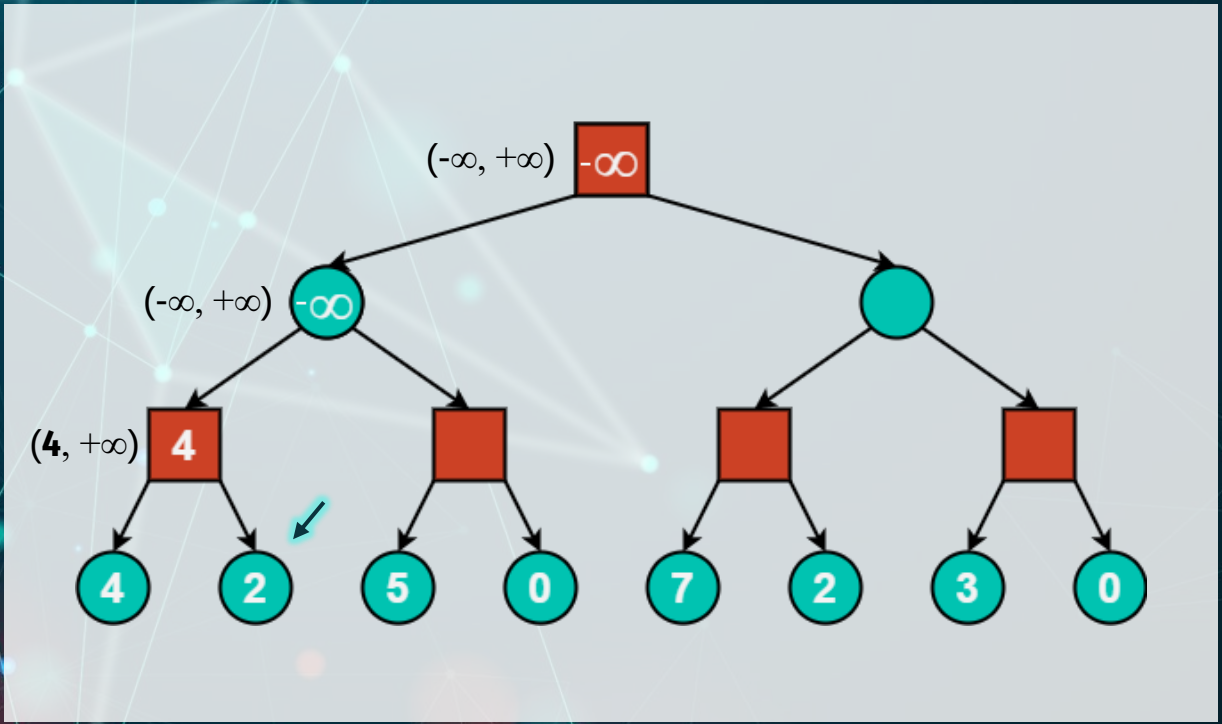
Zadatak 5 – Rešenje



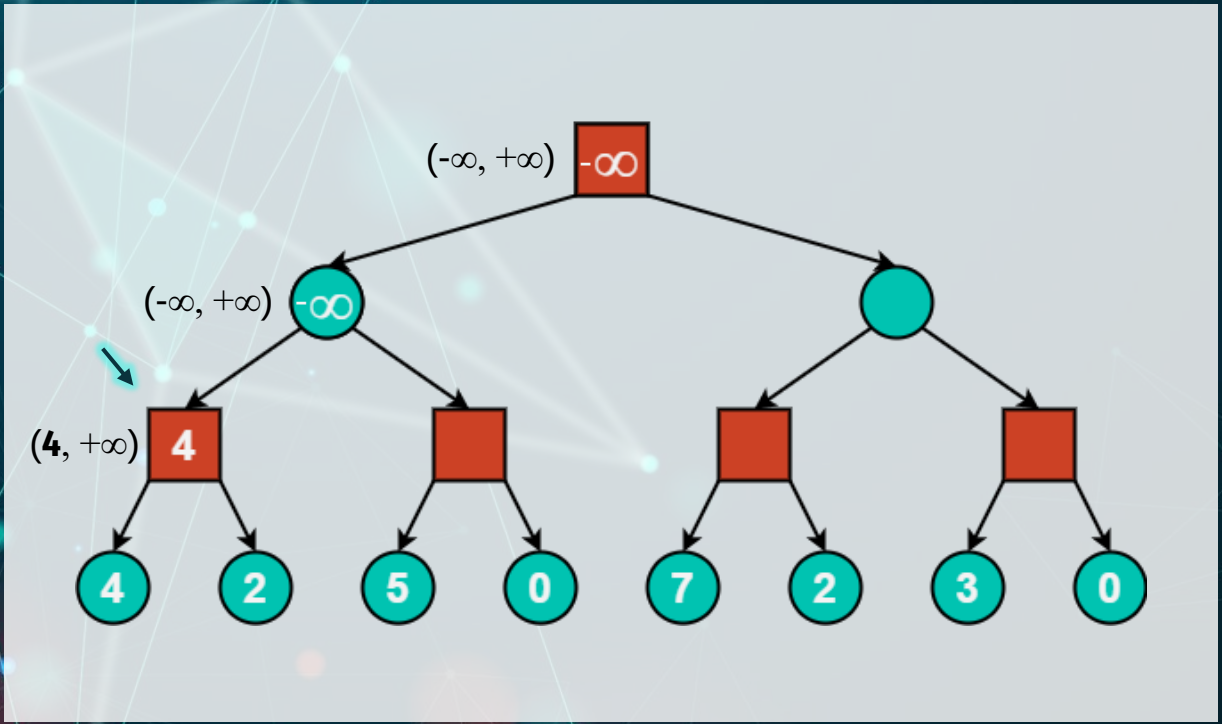
Zadatak 5 – Rešenje



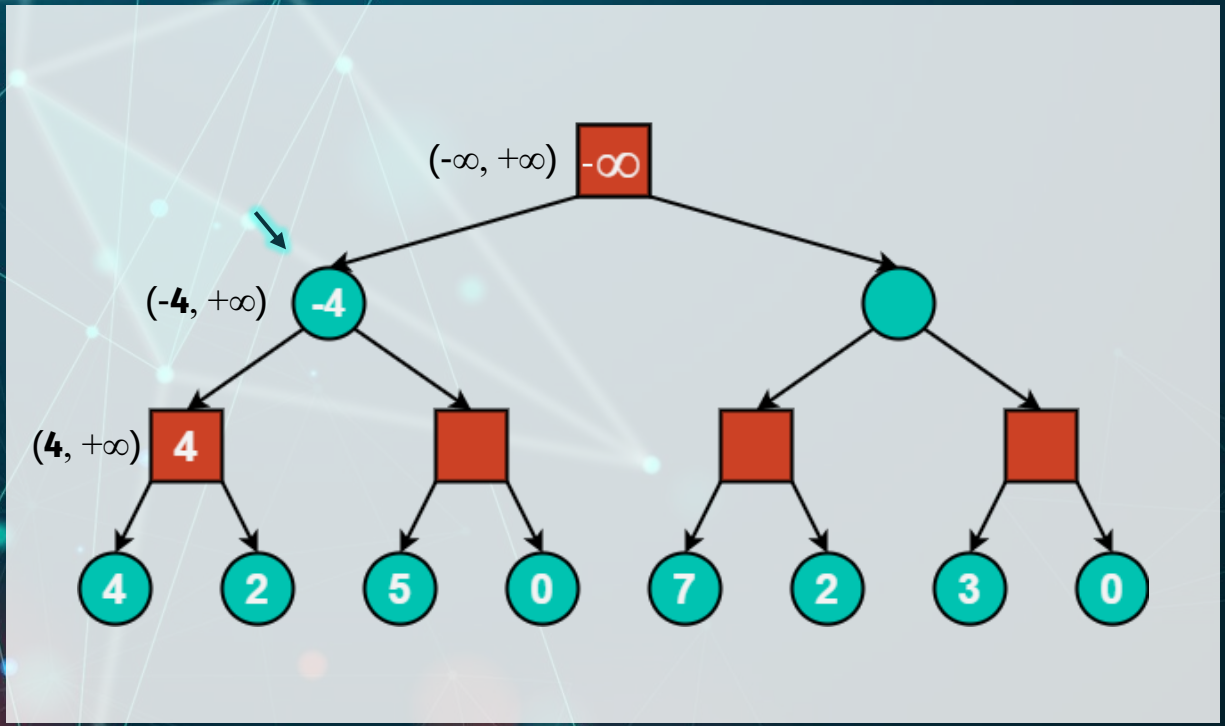
Zadatak 5 – Rešenje



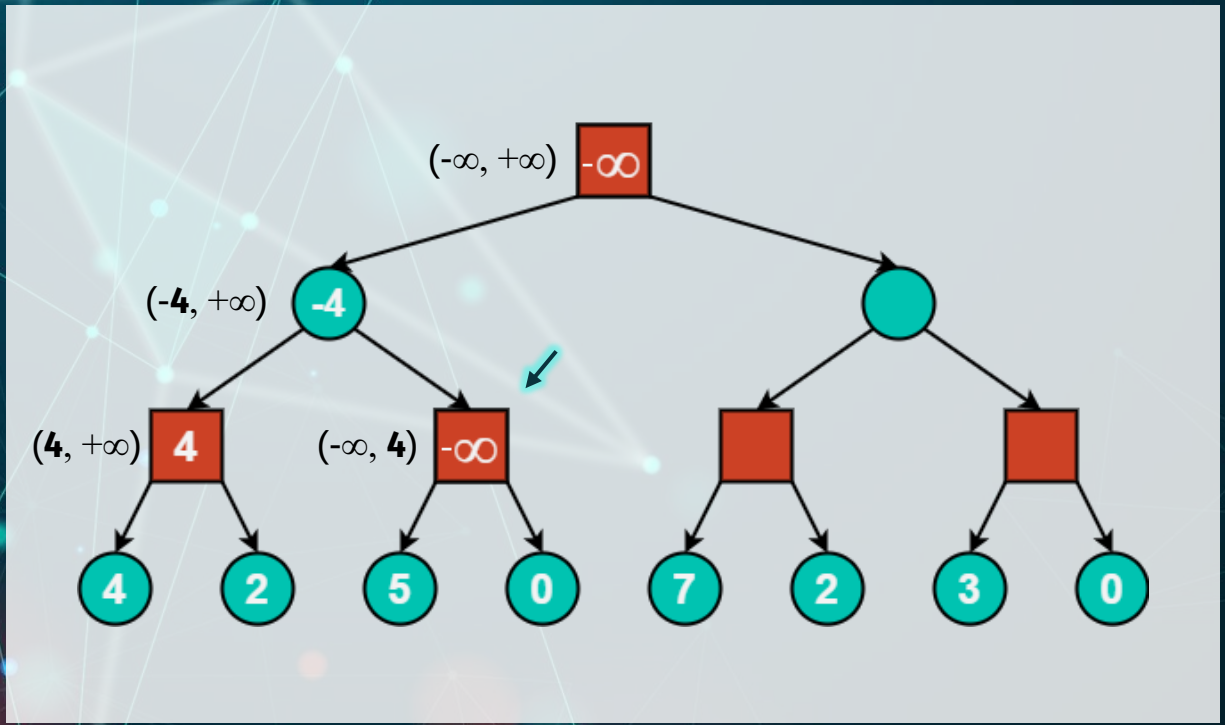
Zadatak 5 – Rešenje



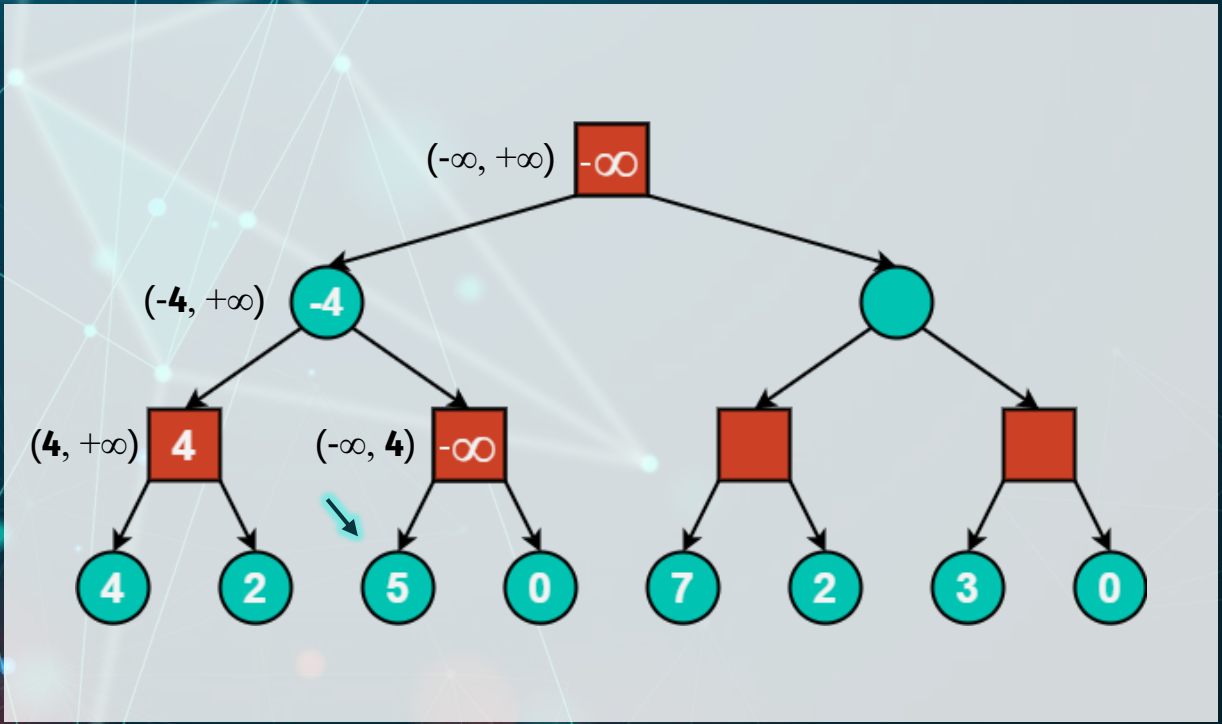
Zadatak 5 – Rešenje



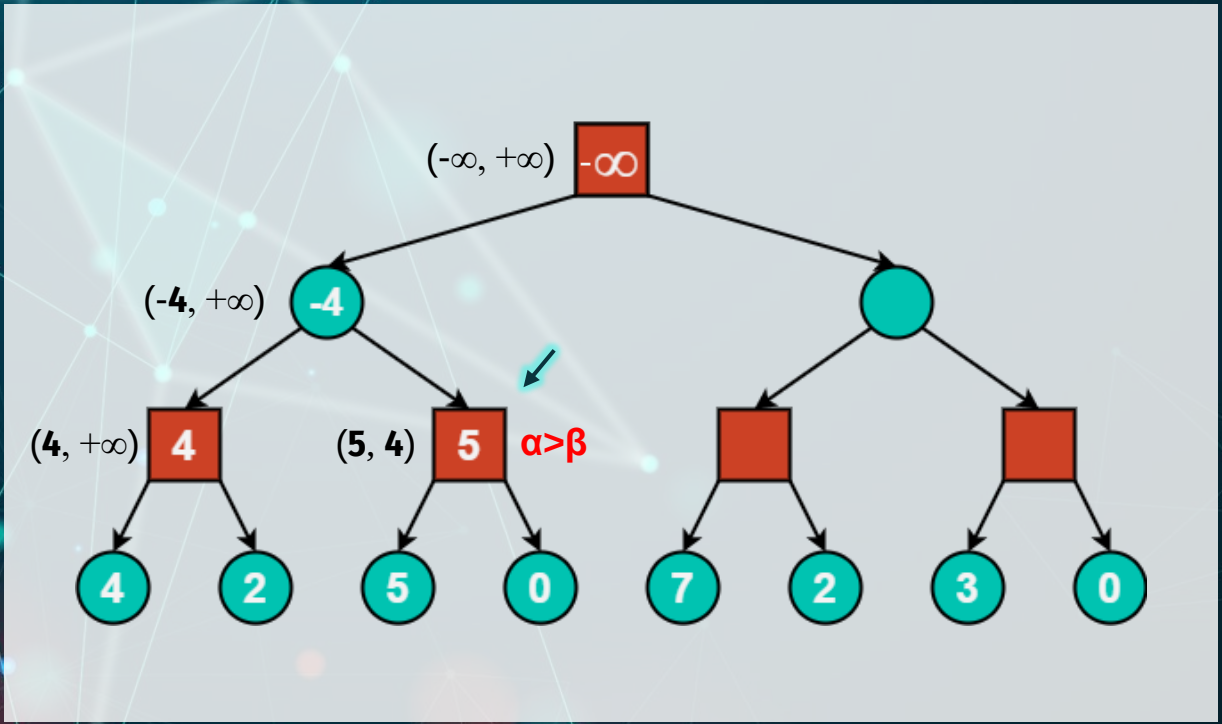
Zadatak 5 – Rešenje



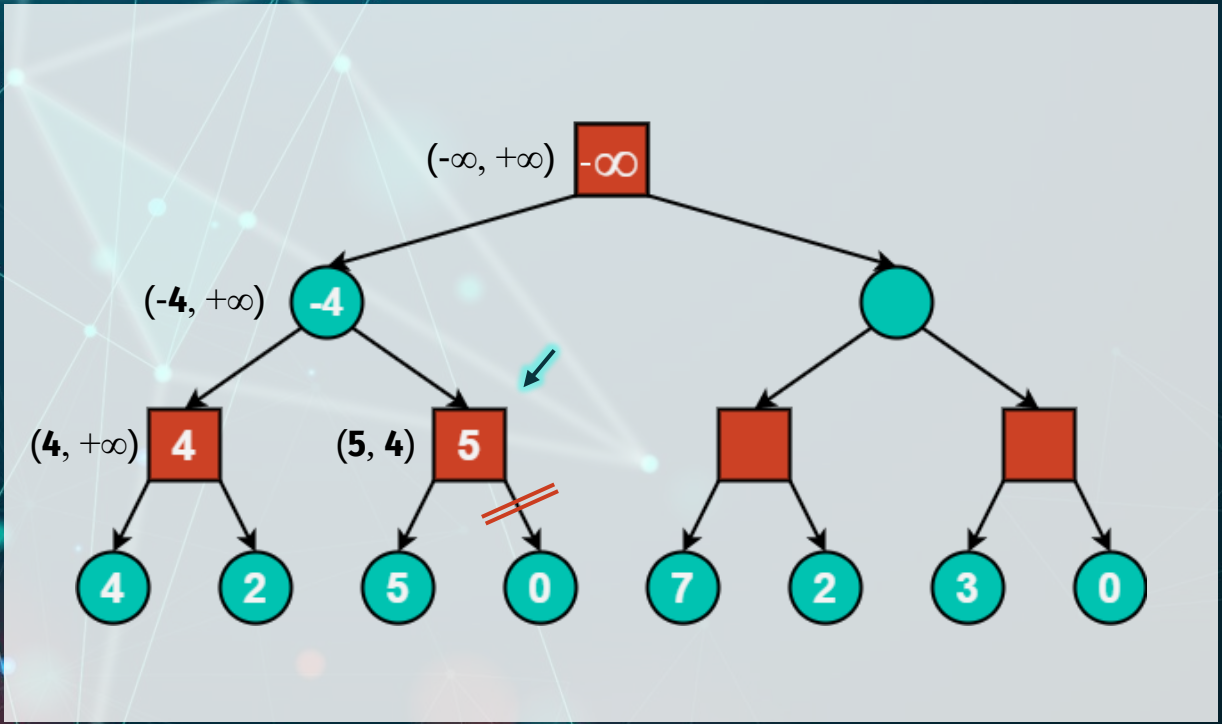
Zadatak 5 – Rešenje



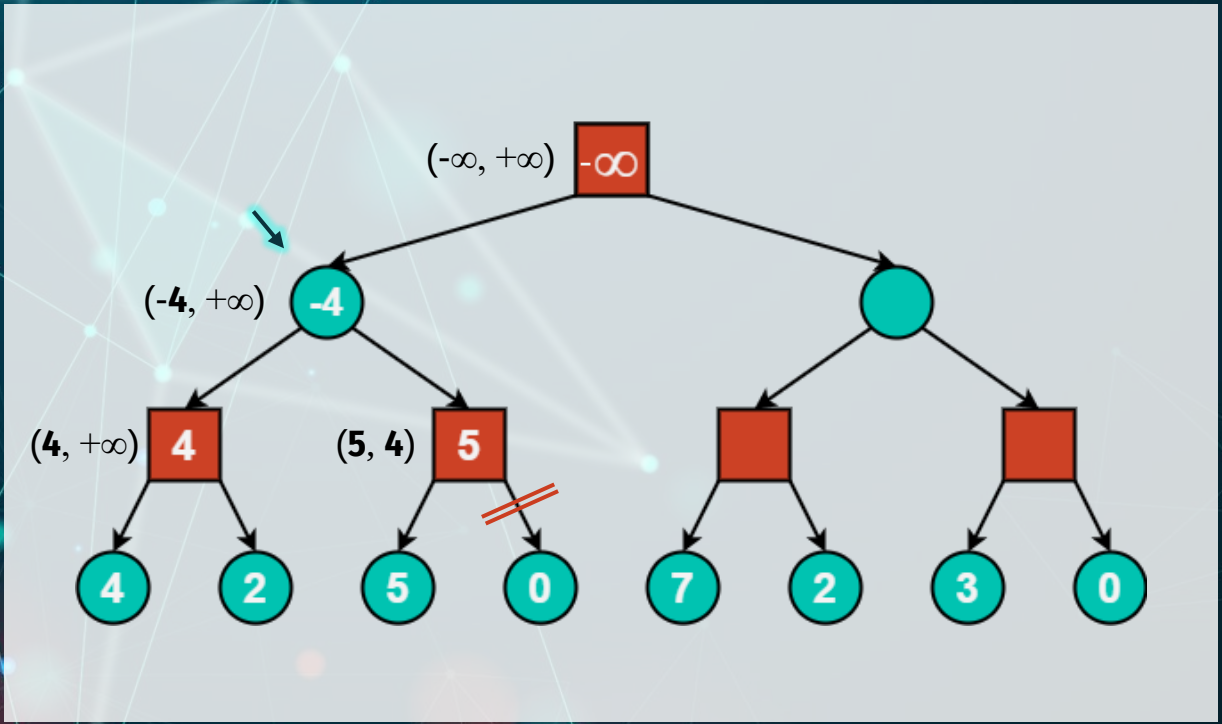
Zadatak 5 – Rešenje



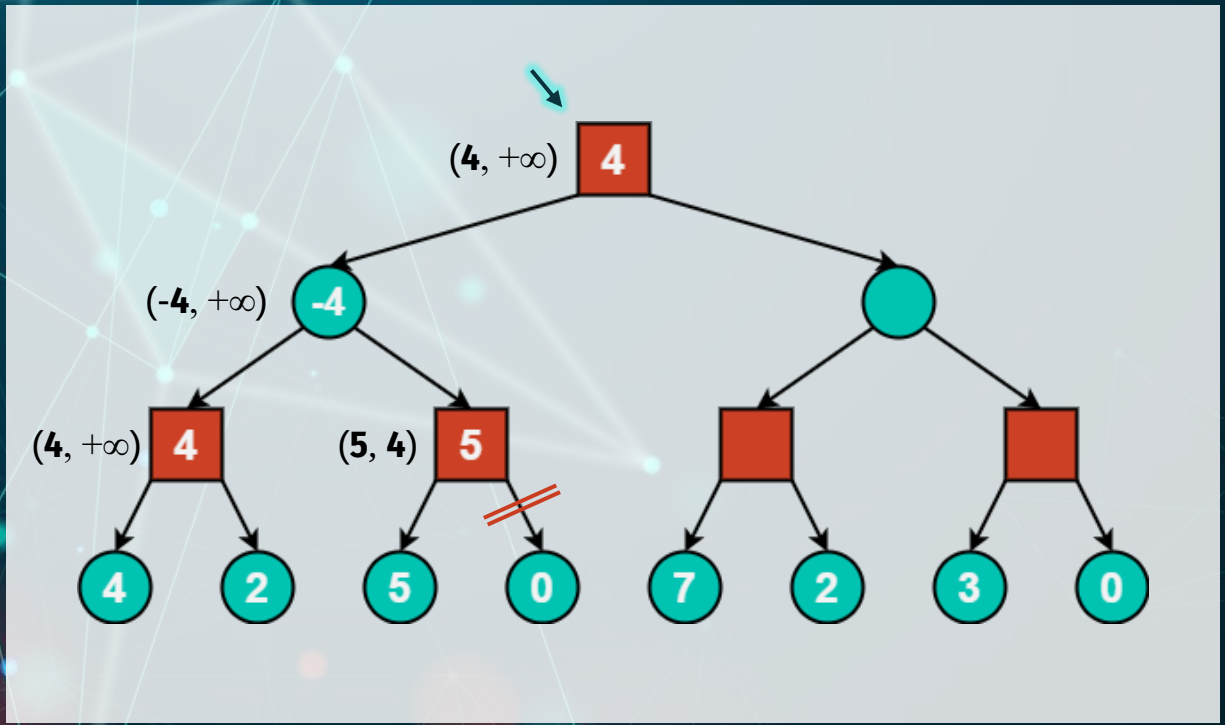
Zadatak 5 – Rešenje



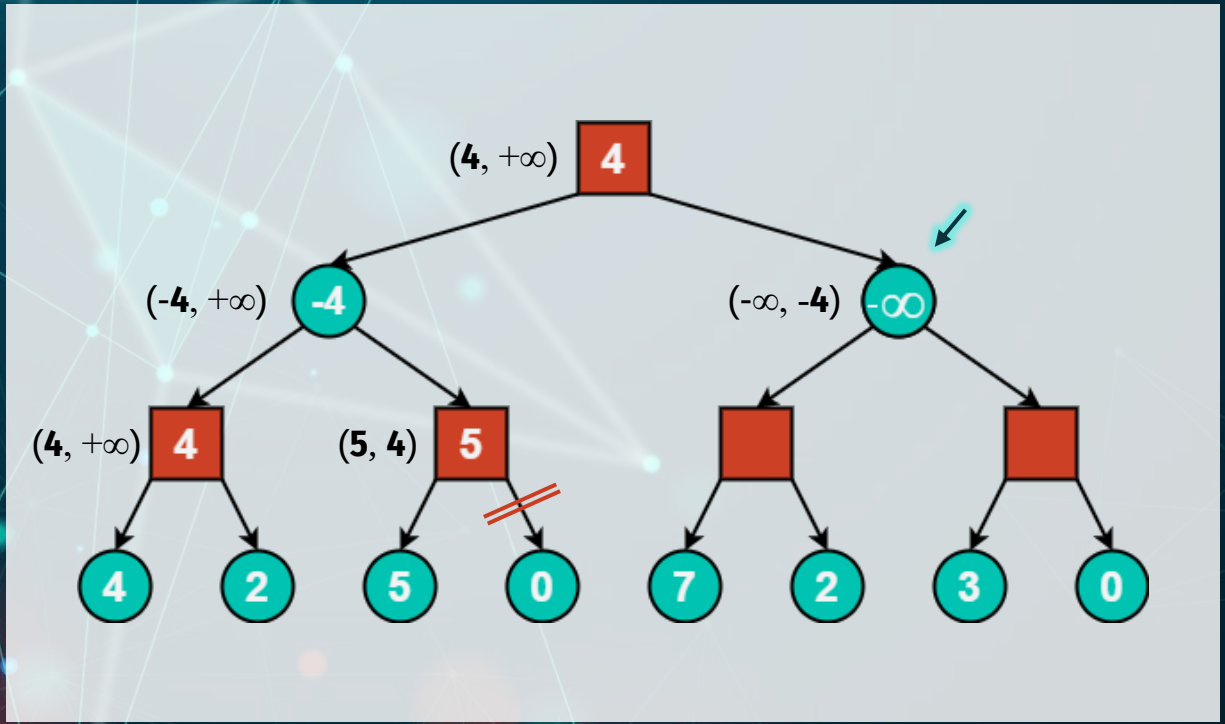
Zadatak 5 – Rešenje



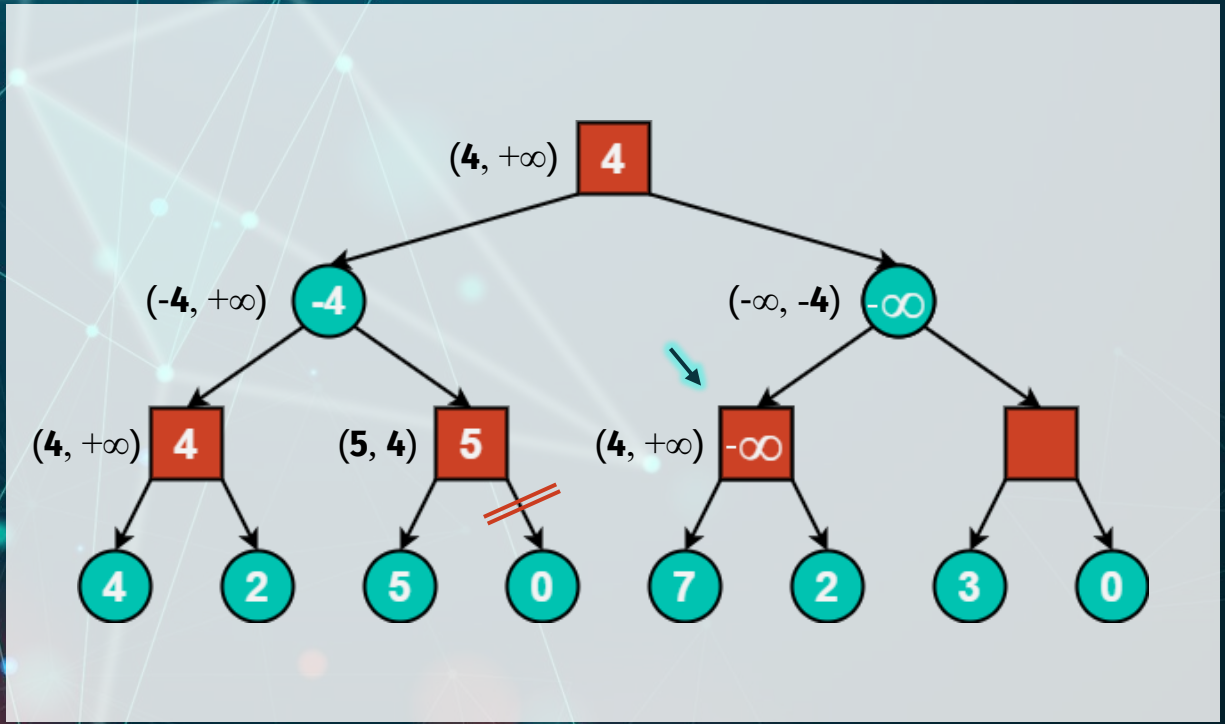
Zadatak 5 – Rešenje



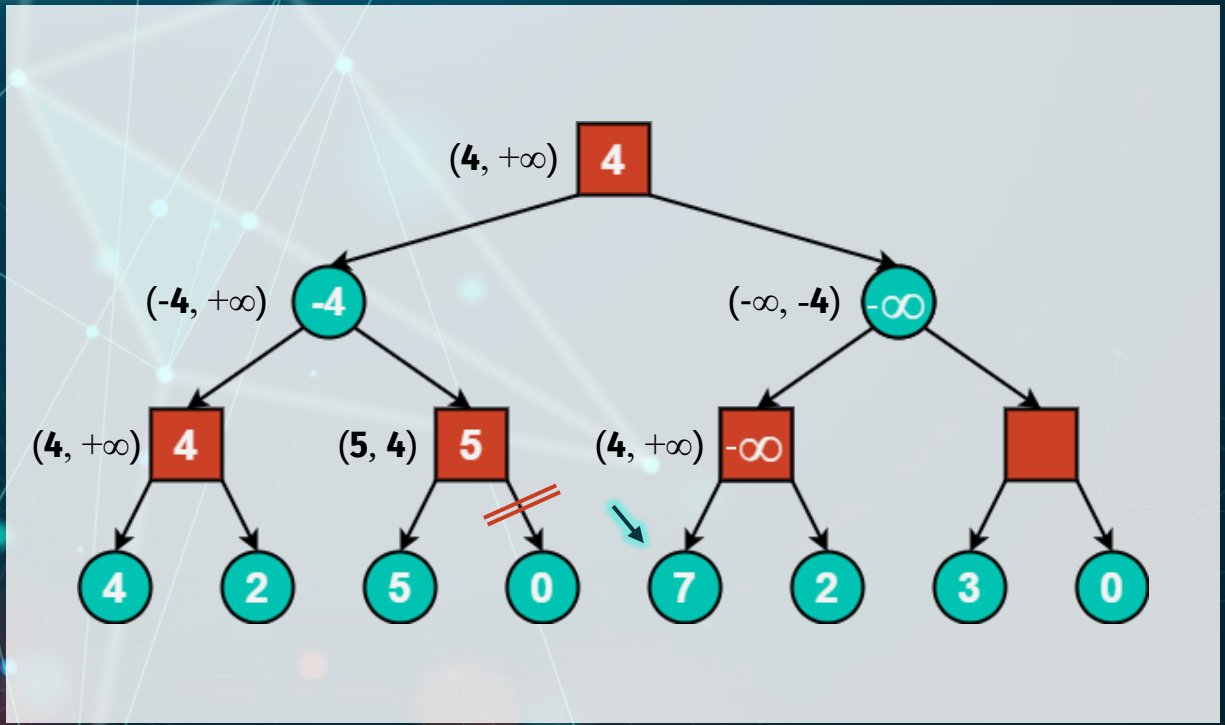
Zadatak 5 – Rešenje



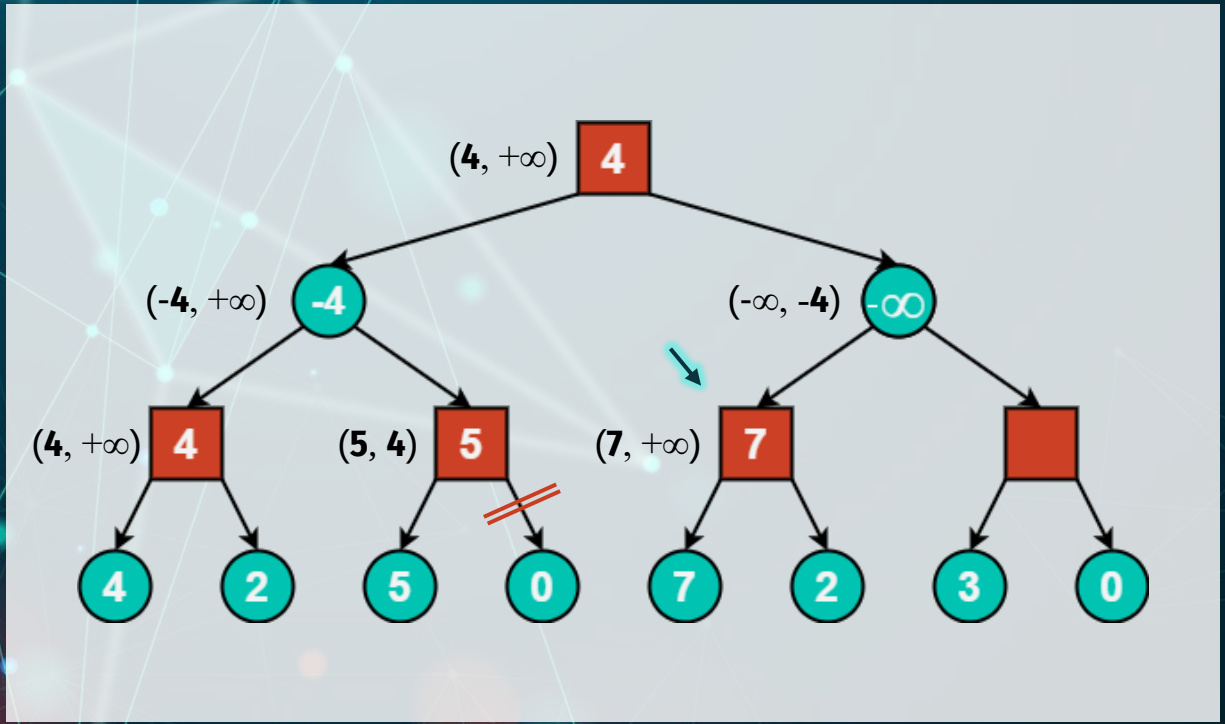
Zadatak 5 – Rešenje



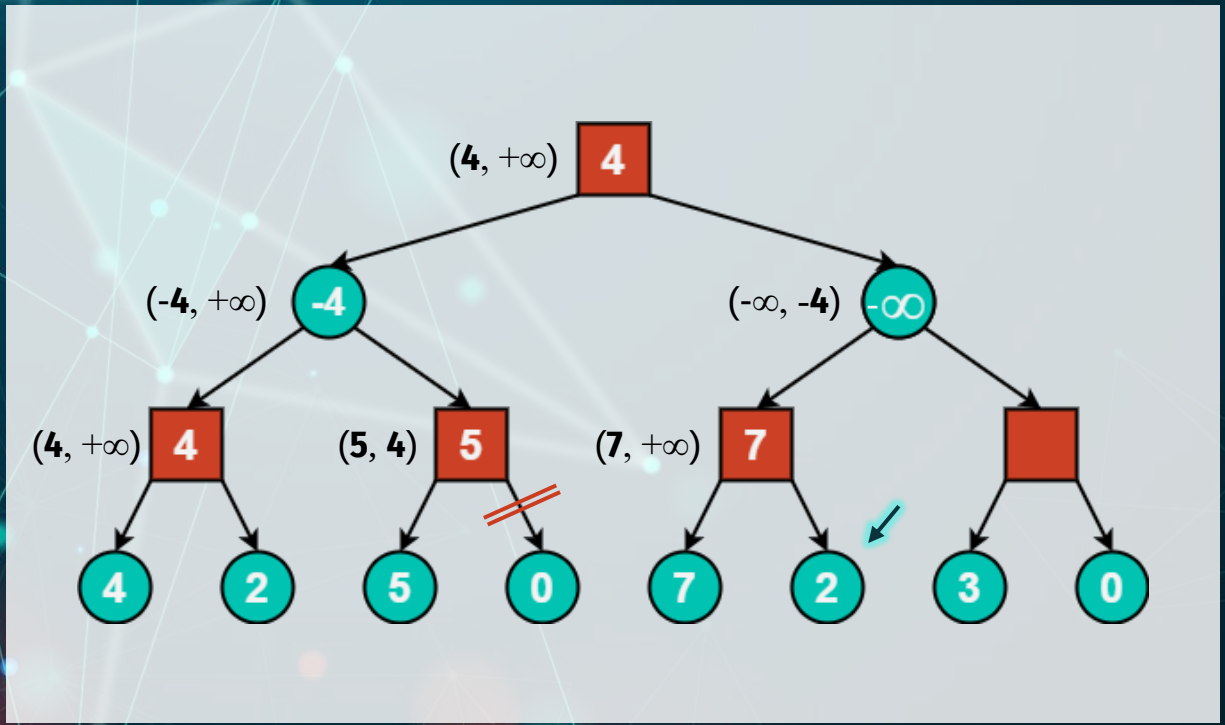
Zadatak 5 – Rešenje



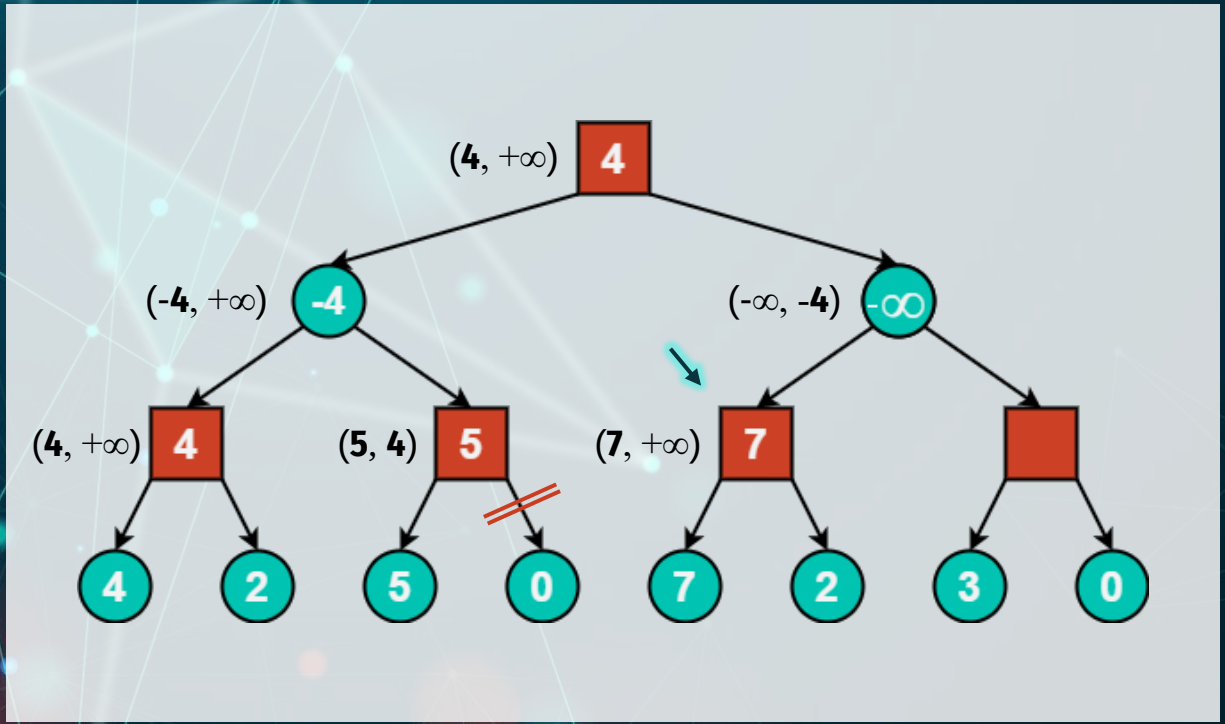
Zadatak 5 – Rešenje



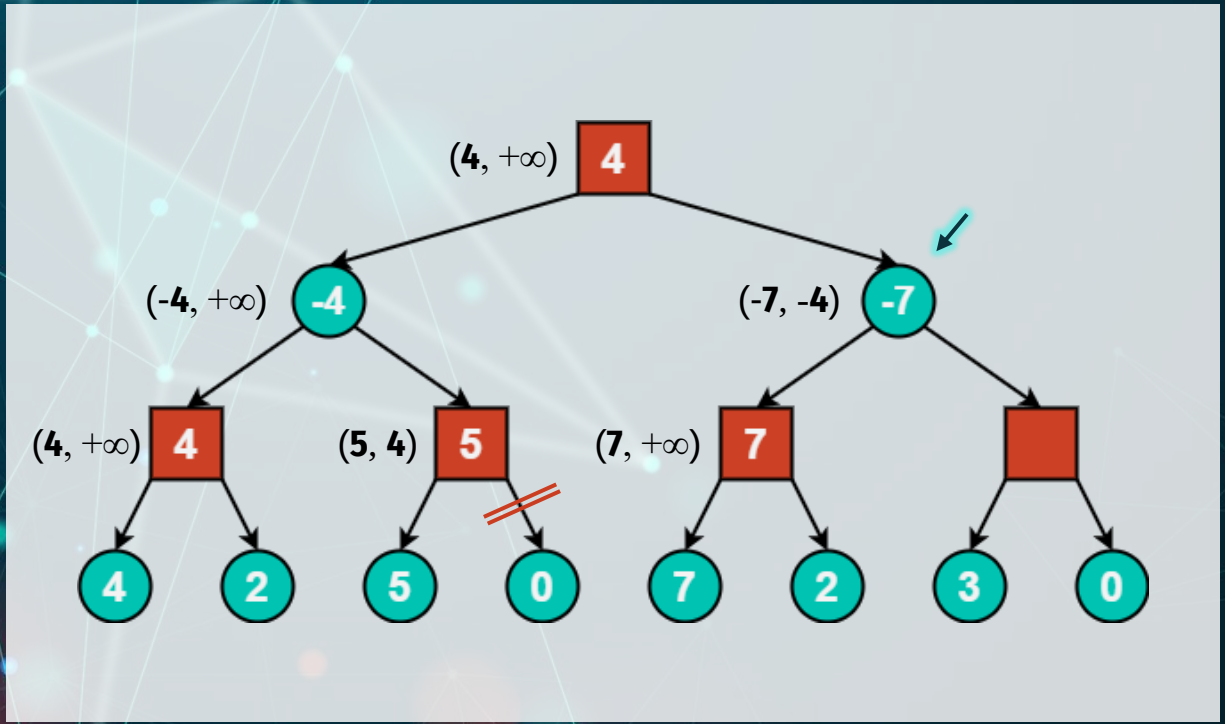
Zadatak 5 – Rešenje



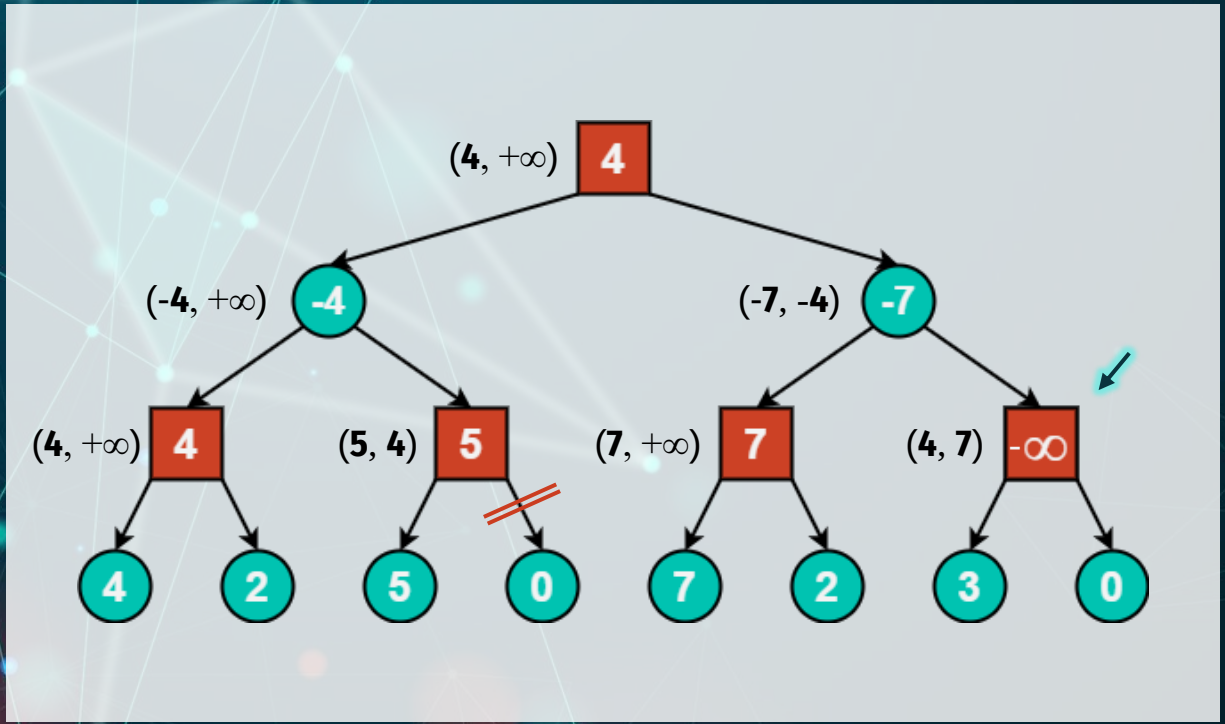
Zadatak 5 – Rešenje



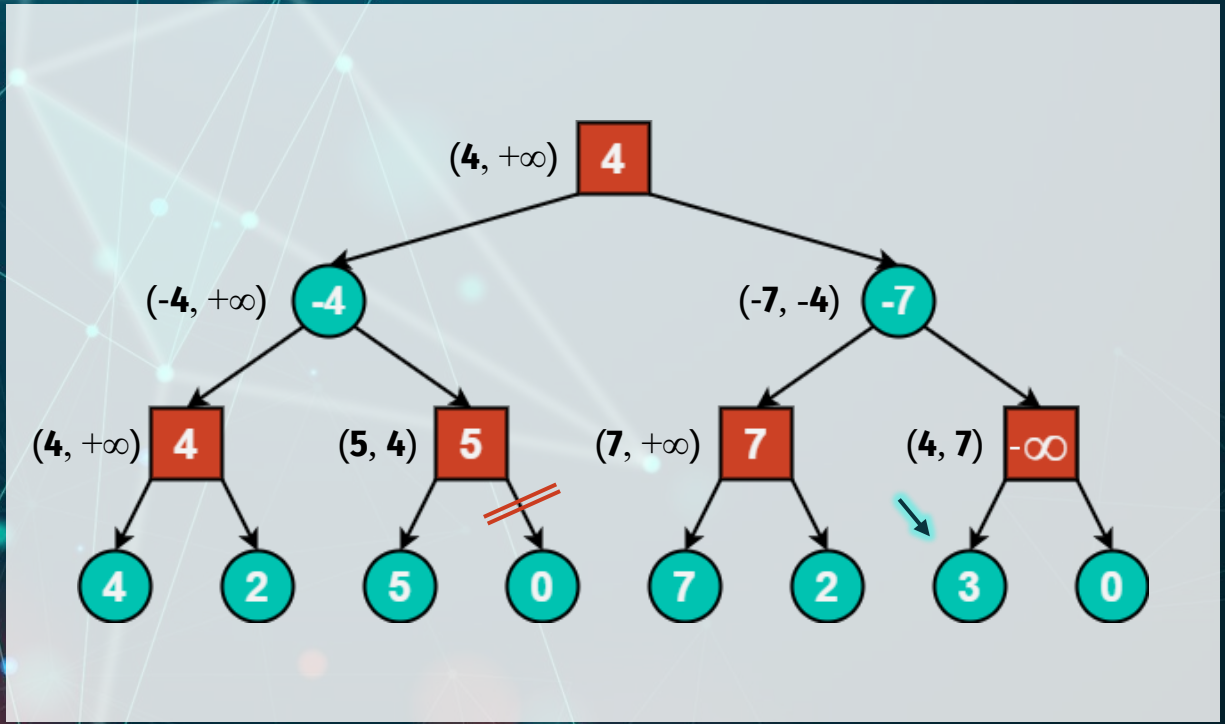
Zadatak 5 – Rešenje



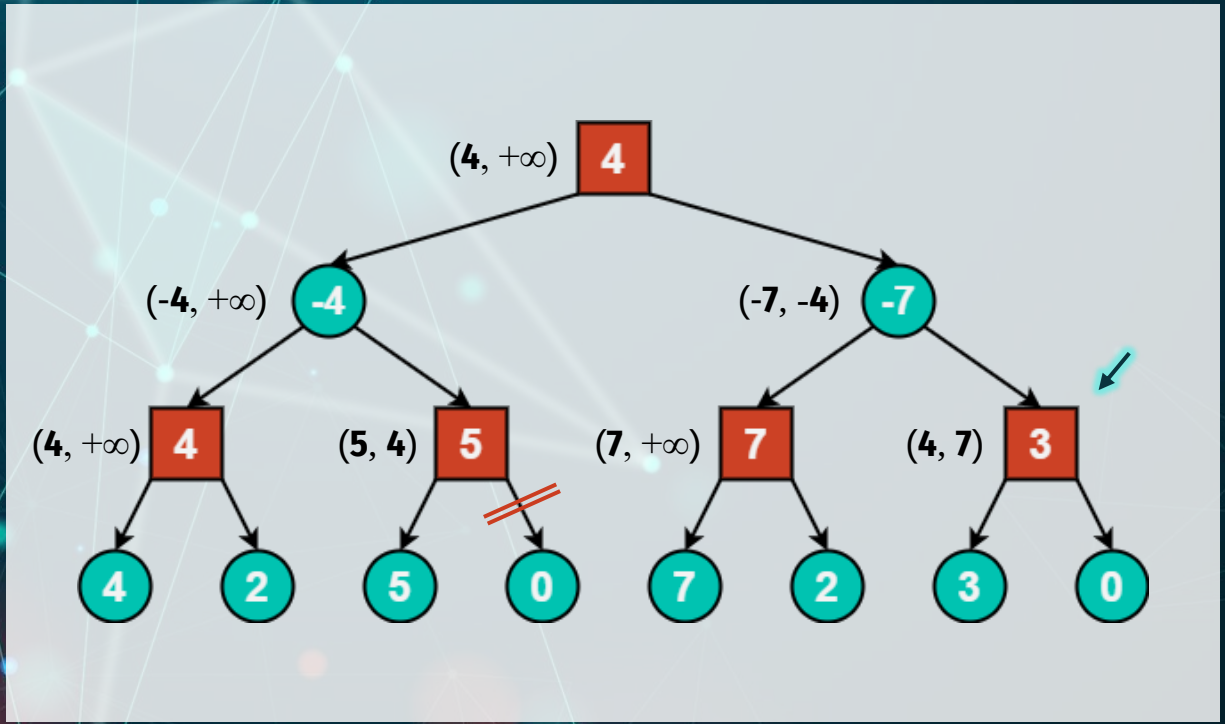
Zadatak 5 – Rešenje



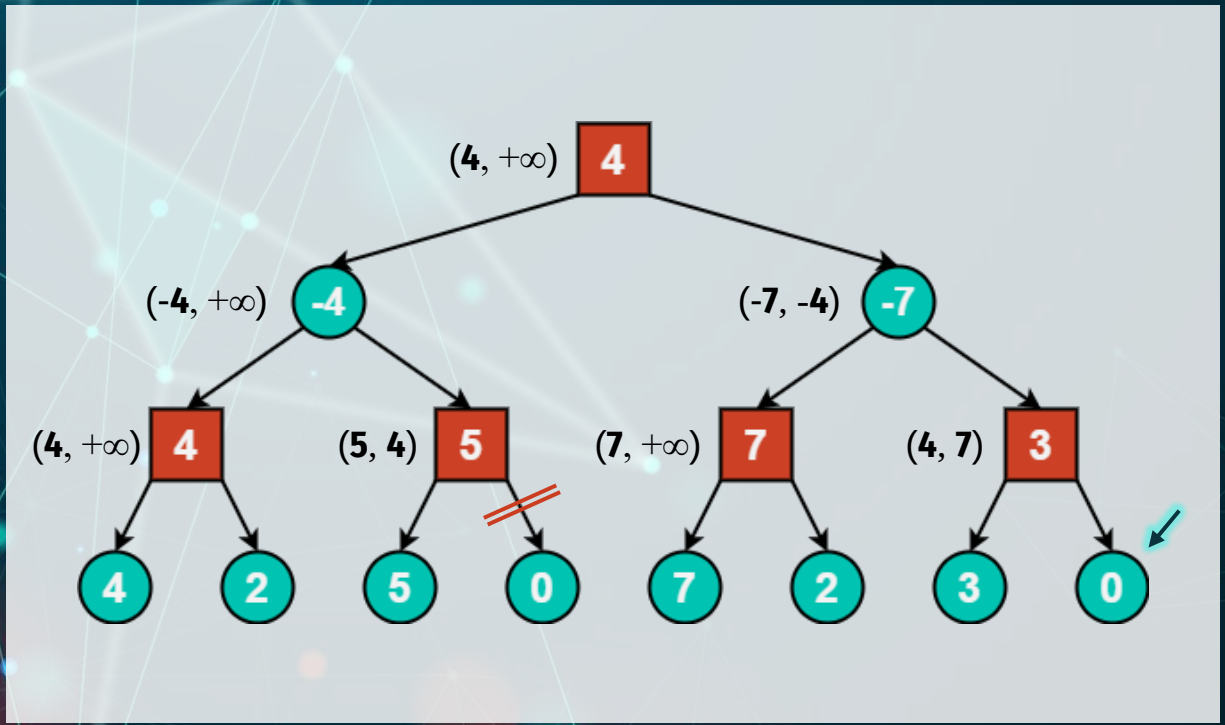
Zadatak 5 – Rešenje



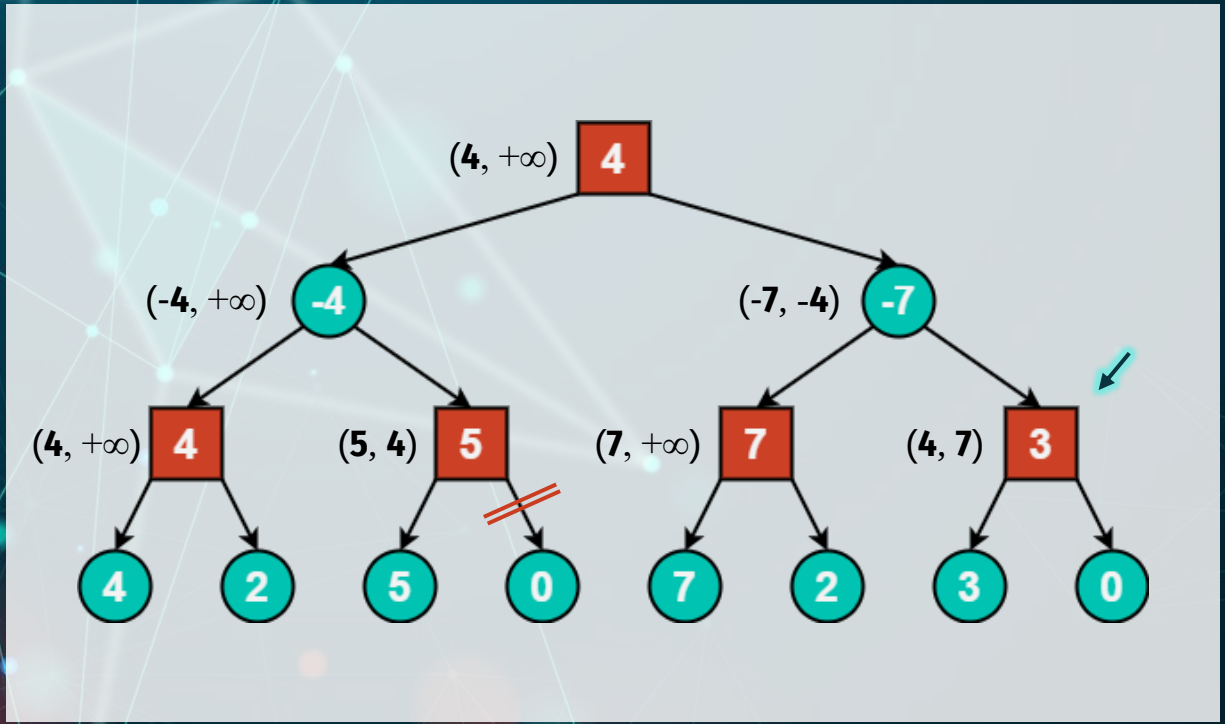
Zadatak 5 – Rešenje



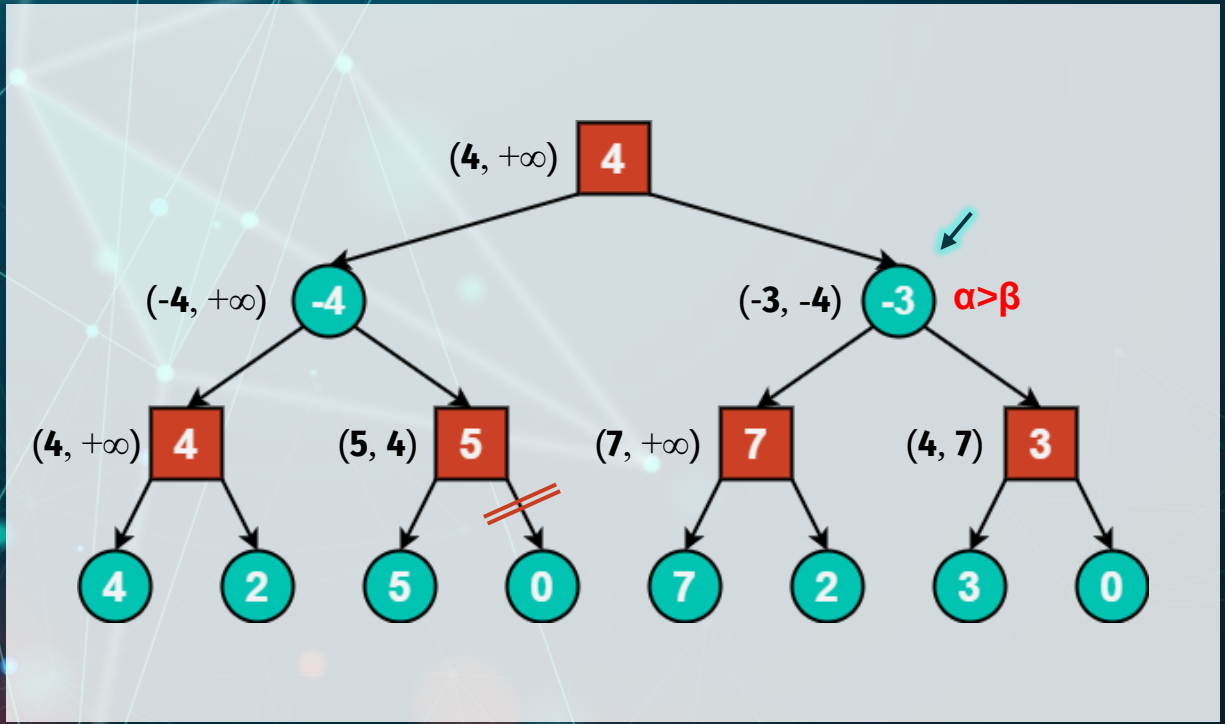
Zadatak 5 – Rešenje



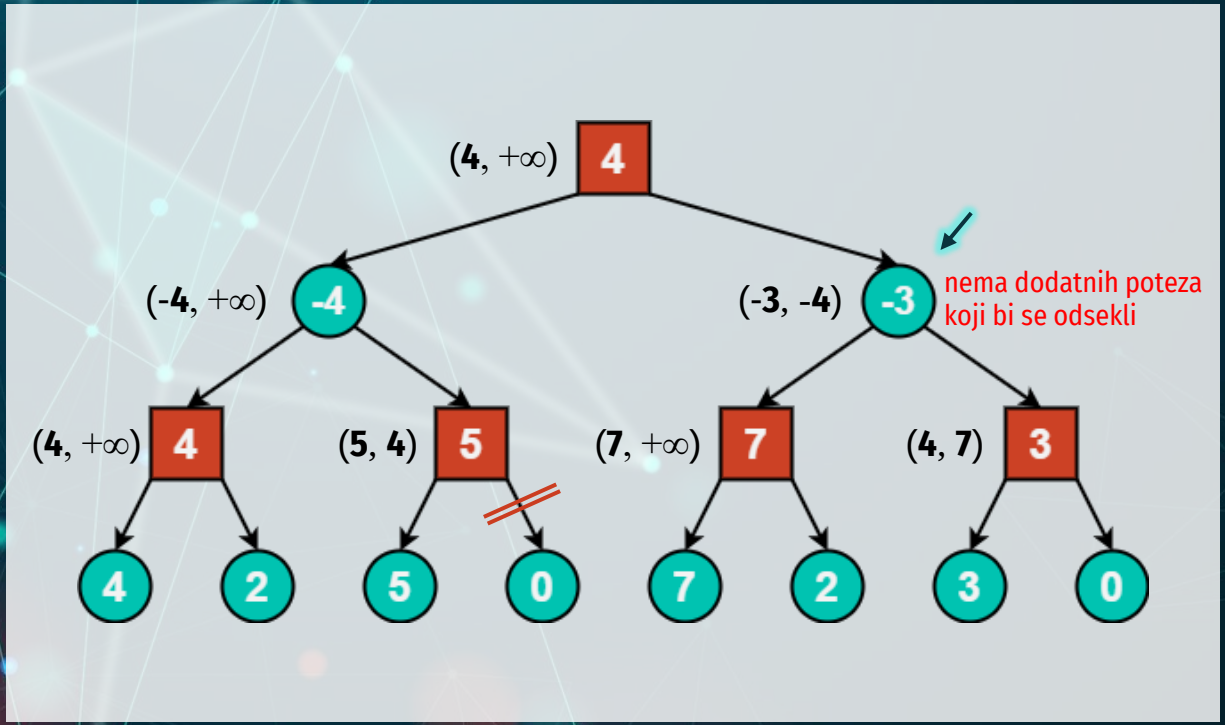
Zadatak 5 – Rešenje



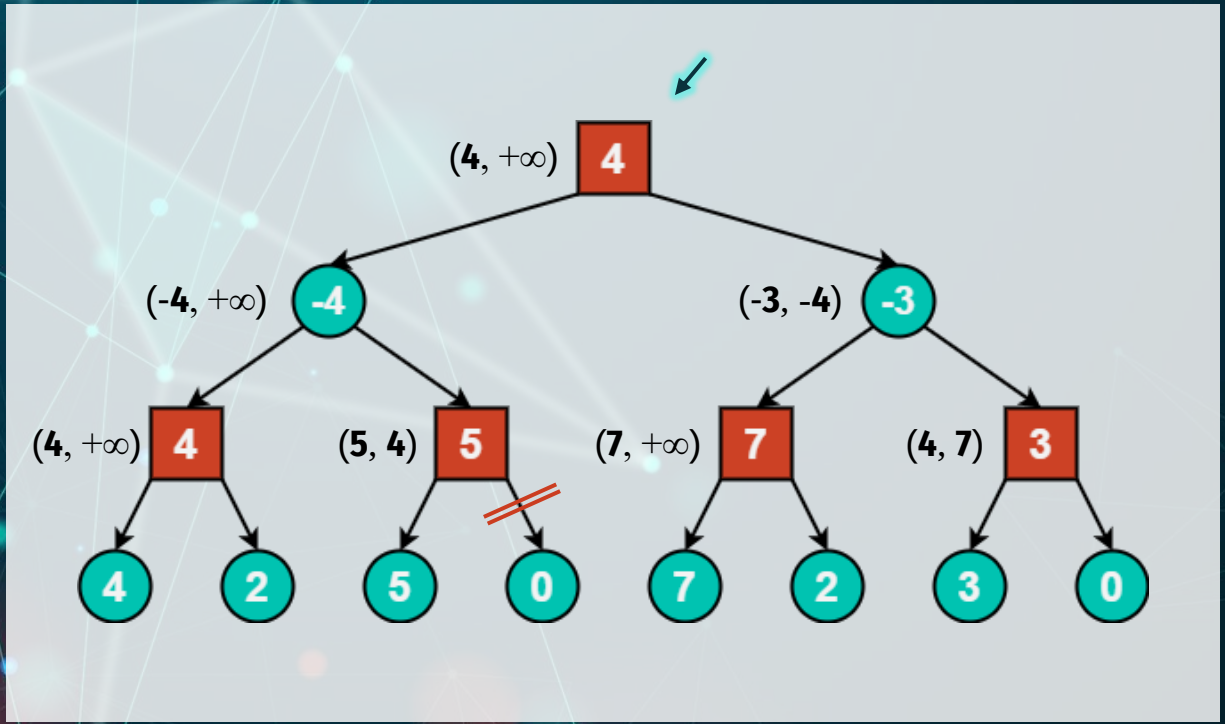
Zadatak 5 – Rešenje



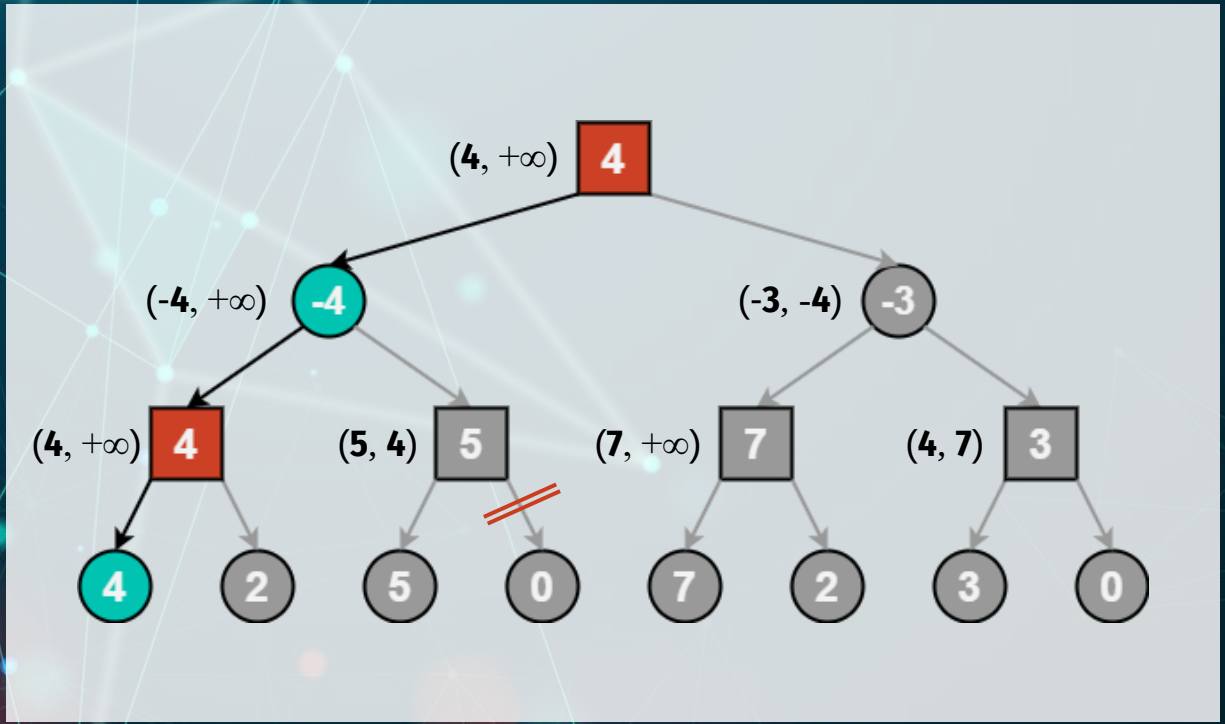
Zadatak 5 – Rešenje



Zadatak 5 – Rešenje



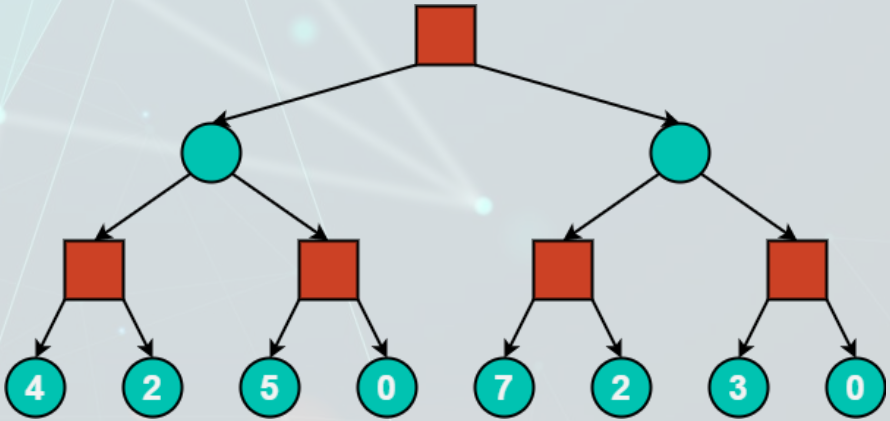
Zadatak 5 – Rešenje



Zadatak 6 – Negascout

★★★★★

Dato je kompletno stablo igre sa naznačenim dobicima za prvog igrača. Primenom **negascout** algoritma odrediti funkciju procene za svaki čvor, konačan ishod igre i sekvencu poteza koja će biti odigrana do kraja ukoliko su oba igrača racionalna.



NEGASCOUT IDEJA

Negascout predstavlja unapređenje algoritma alfa-beta odsecanja. Negascout algoritam sigurno neće razmatrati čvorove koje ne razmatra ni alfa-beta odsecanje, a može dodatno smanjiti broj čvorova koje bi inače algoritam alfa-beta odsecanja razmatrao.

Ideja negascout algoritma je da se u punom prozoru [alpha, beta] vrši evaluacija prve pozicije naslednika (za koju se tvrdi da je najbolja) kako bi se ustanovile granice prozora, a zatim da se potvrdi da su ostale pozicije gore od prve koristeći prozor nulte veličine [-alpha-1, -alpha]. Ukoliko se za neku poziciju utvrdi da povećava vrednost alpha onda ta pozicija nije gora od prve pozicije i ta pozicija se mora ponovo evaluirati koristeći prozor pune veličine.

NEGASCOUT IDEJA

Performanse negascout algoritma veoma zavise od redosleda razmatranja pozicija naslednika tekuće pozicije. Ukoliko je prva pozicija naslednik koja se razmatra ujedno i najbolja od ostalih onda negascout vrši više odsecanja nego običan alfa-beta algoritam. U tom slučaju su performanse bolje za 10-20%.

Određivanje najboljeg naslednika može se izvesti izborom dobre heuristike ili prethodnom običnom plitkom minimax pretragom.

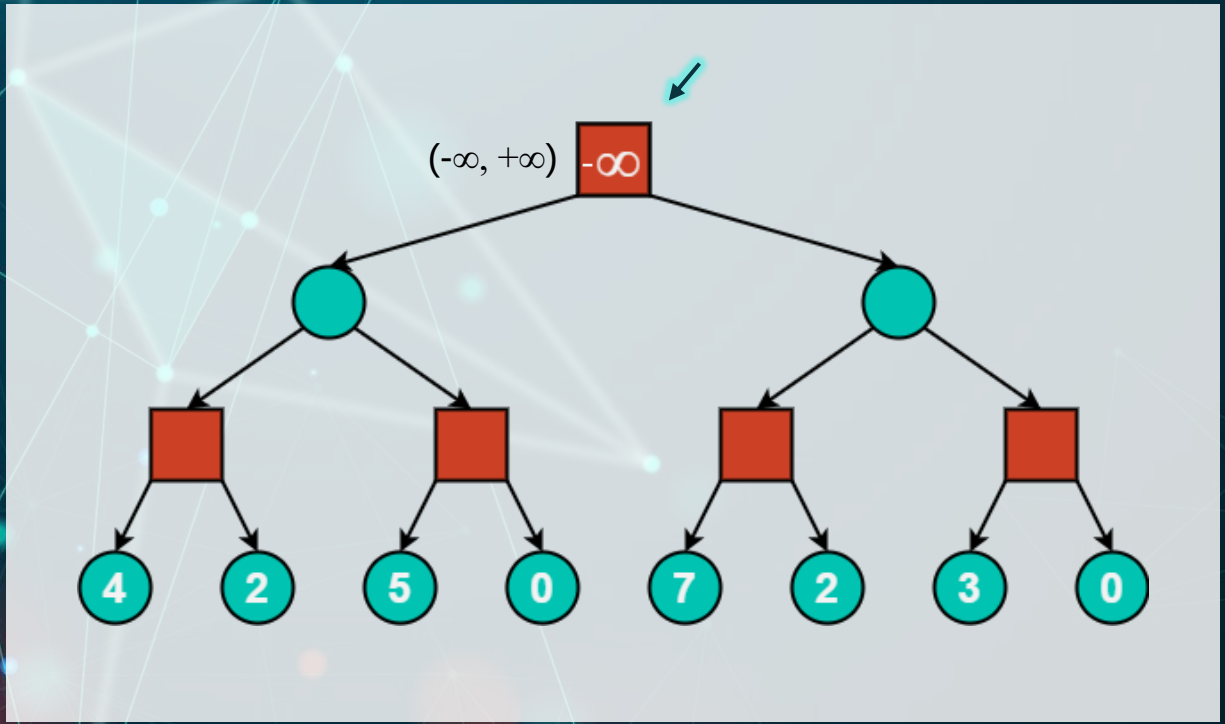
Ukoliko se izbor pozicije naslednika tekuće pozicije, kao prve sledeće pozicije koja se razvija, vrši na slučajan način onda se ovaj algoritam sporije izvršava u odnosu na običan algoritam alfa-beta odsecanja. Iako i dalje neće razmatrati pozicije koje ne razmatra ni alfa-beta odsecanje moraće da razmatra iste pozicije više puta.

NEGASCOUT ALGORITAM

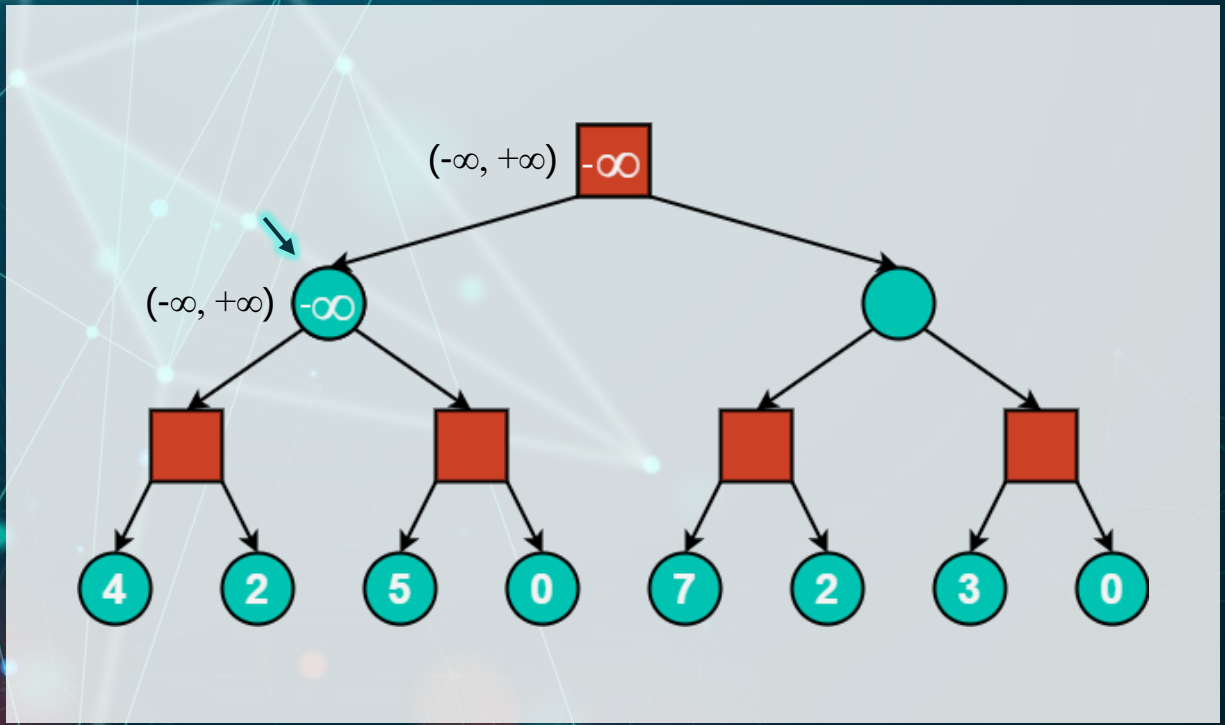
```
def negascout(node, player, alpha, beta):
    if is_terminal_node(node):
        return node_evaluation(node) * (-1 if player == Player.MIN else 1)

    score = -math.inf
    for succ in node.successors():
        if succ is node.first_child():
            val = -negascout(succ, switch(player), -beta, -alpha)
        else:
            val = -negascout(succ, switch(player), -alpha - 1, -alpha)
            if alpha < val < beta:
                val = -negascout(succ, switch(player), -beta, -alpha)
        score = max(score, val)
        alpha = max(alpha, score)
        if alpha >= beta:
            break
    return score
```

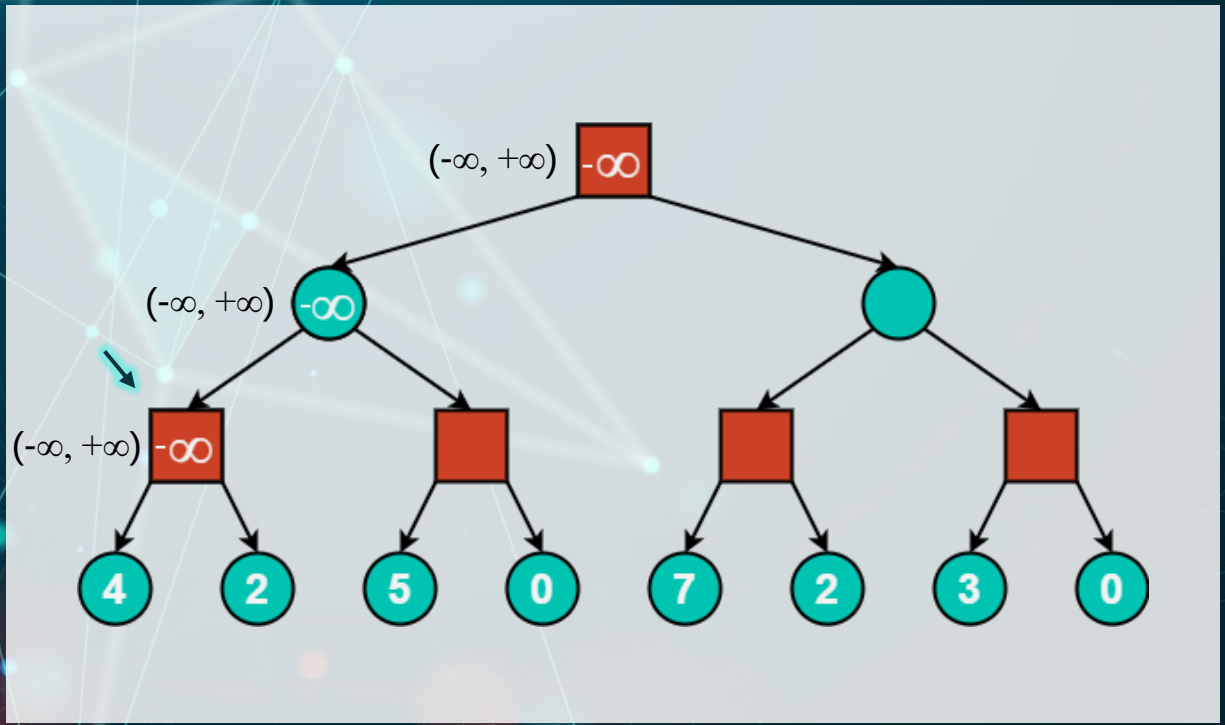
Zadatak 6 – Rešenje



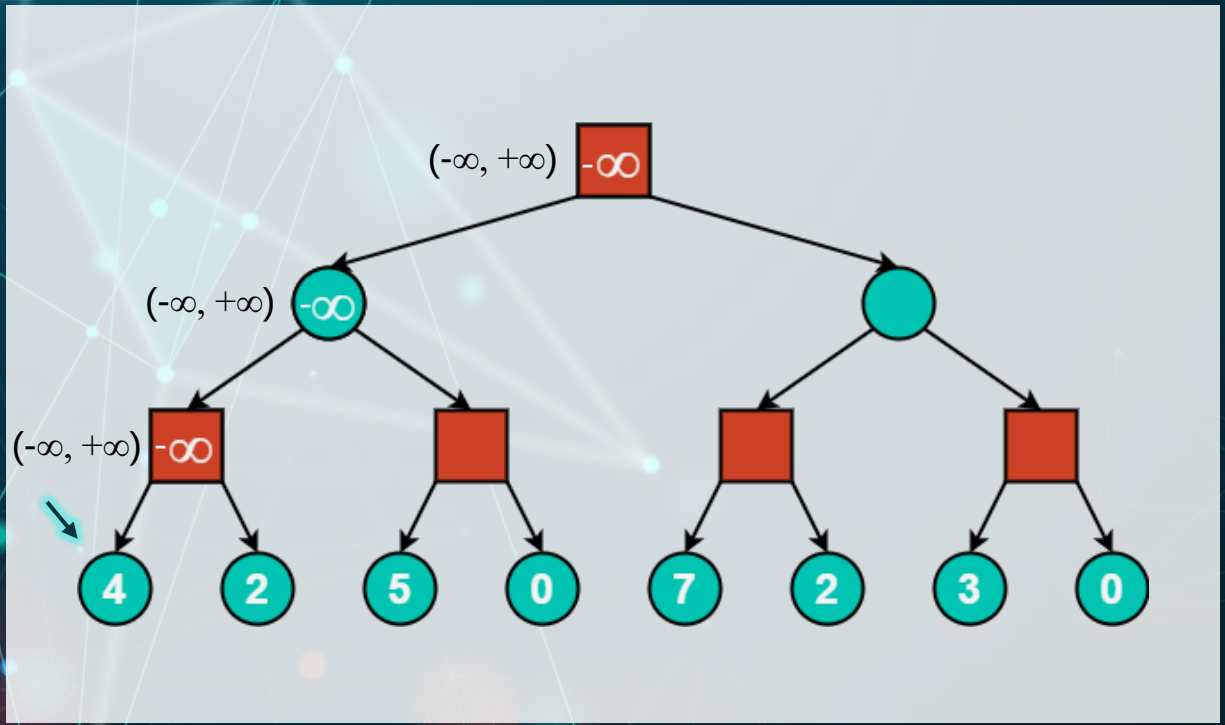
Zadatak 6 – Rešenje



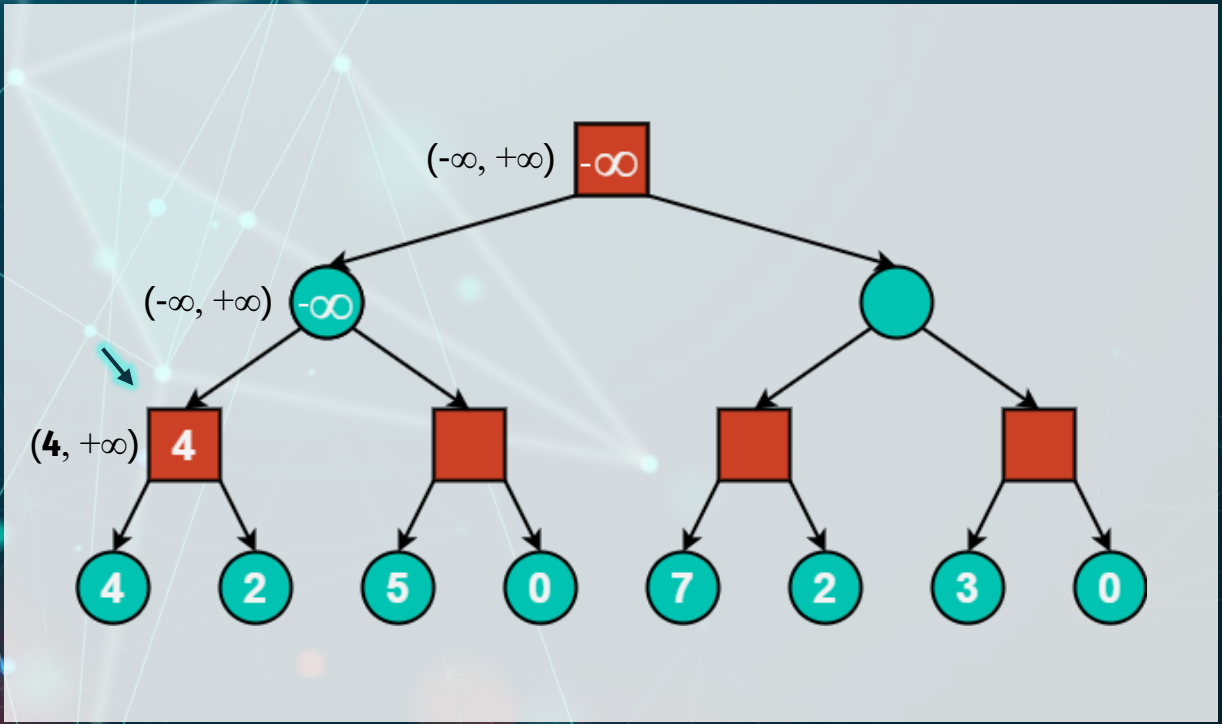
Zadatak 6 – Rešenje



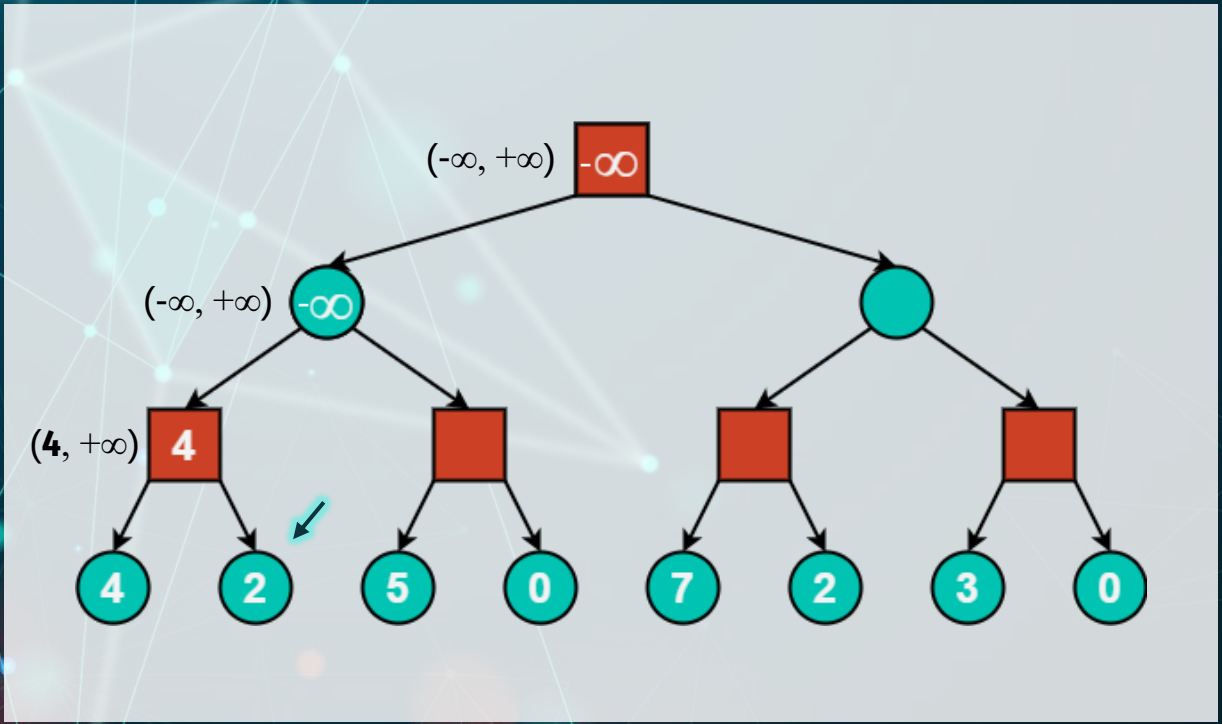
Zadatak 6 – Rešenje



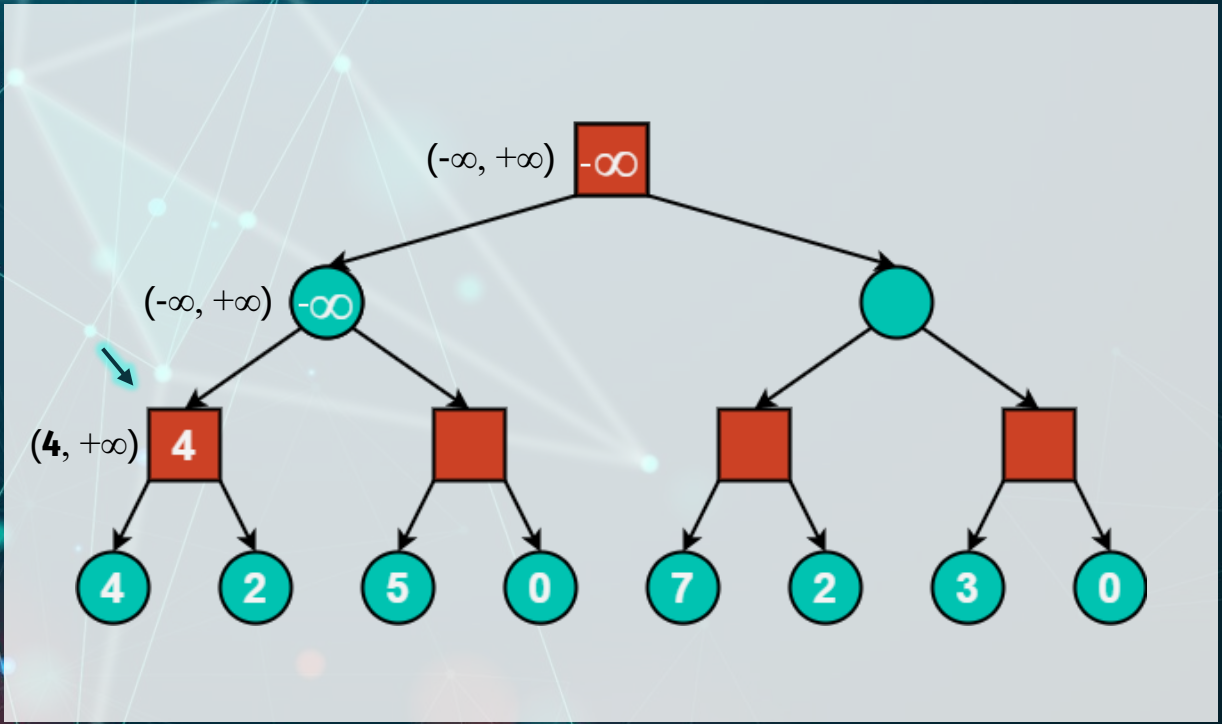
Zadatak 6 – Rešenje



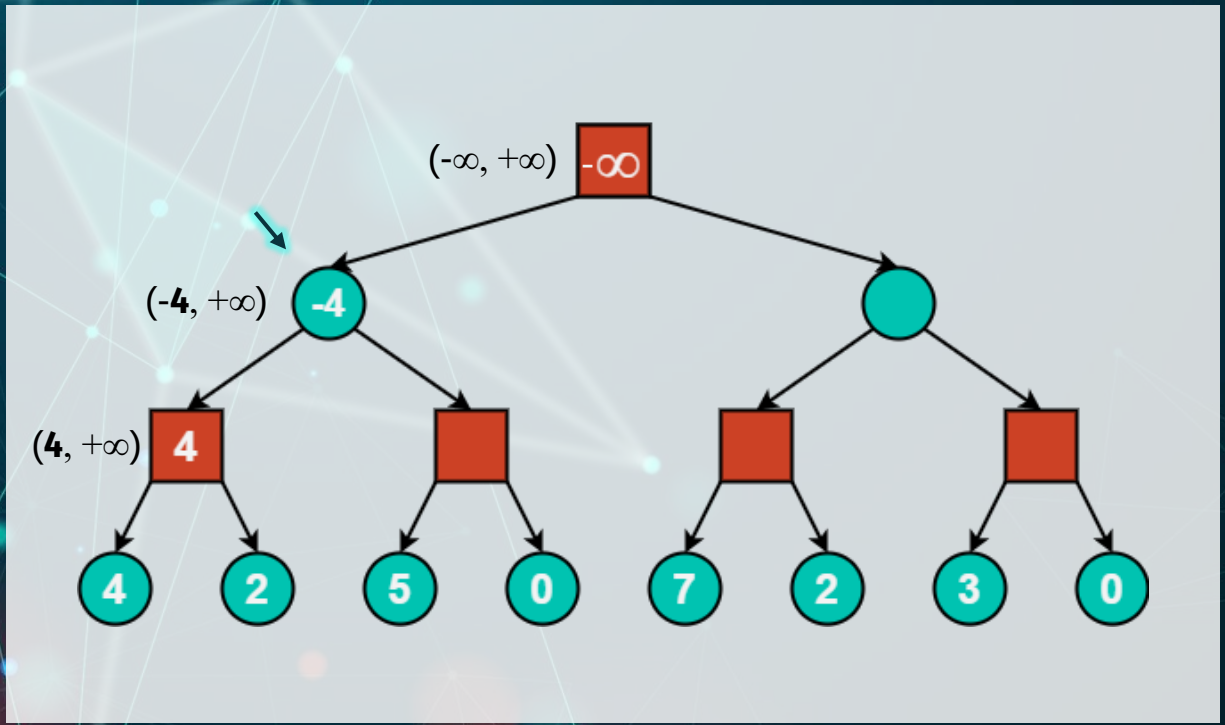
Zadatak 6 – Rešenje



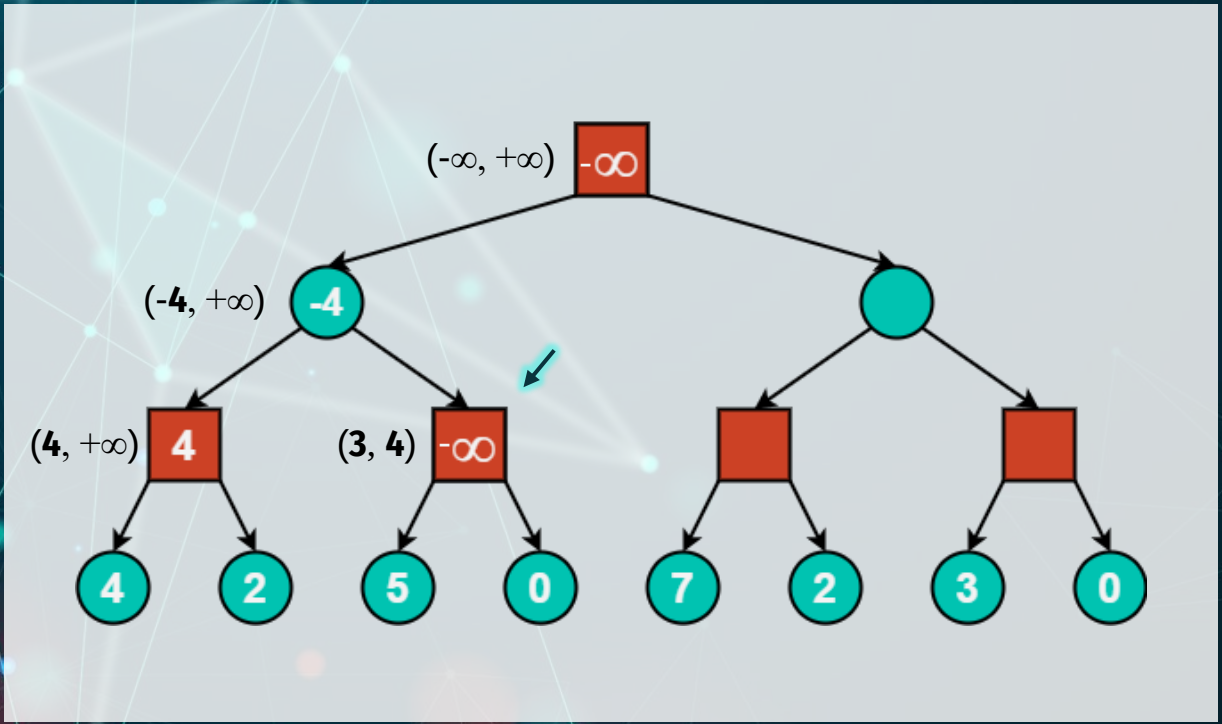
Zadatak 6 – Rešenje



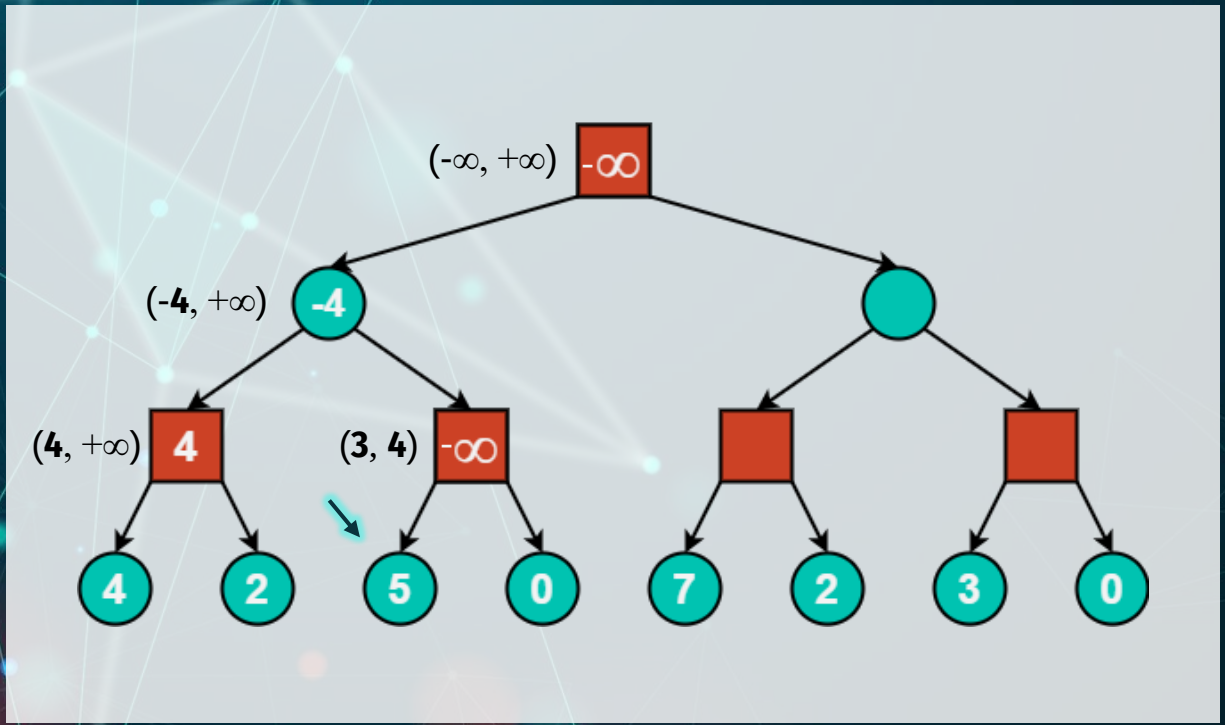
Zadatak 6 – Rešenje



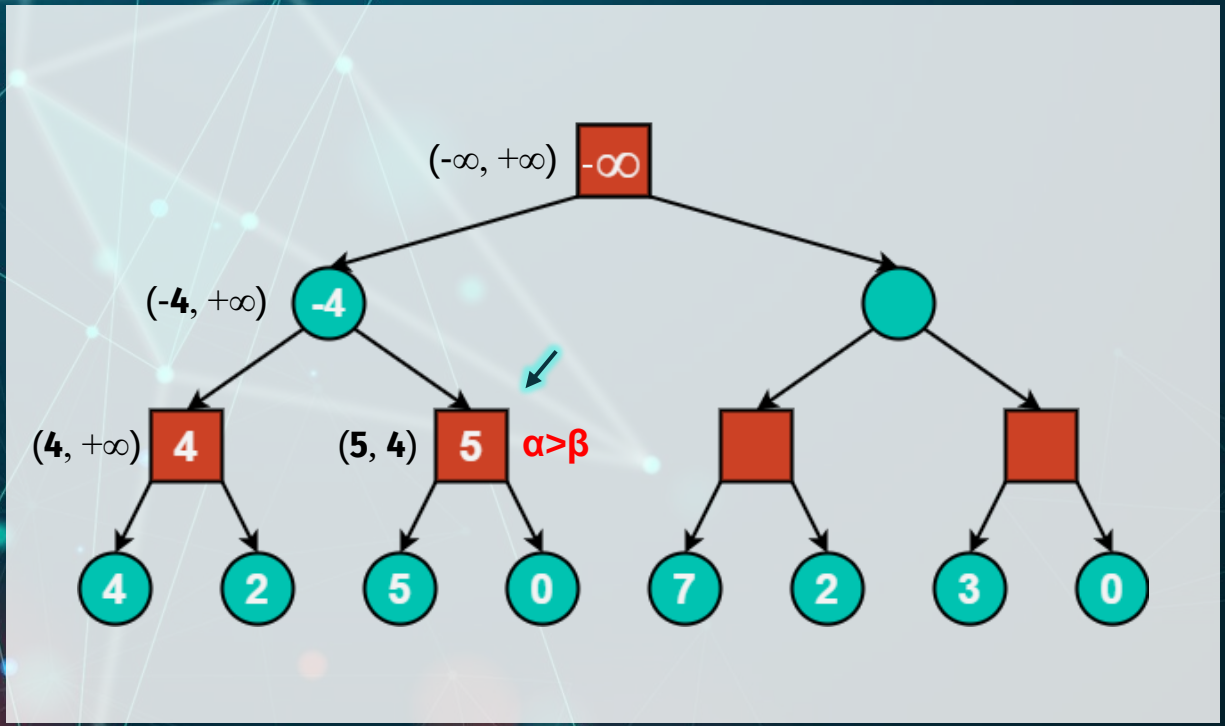
Zadatak 6 – Rešenje



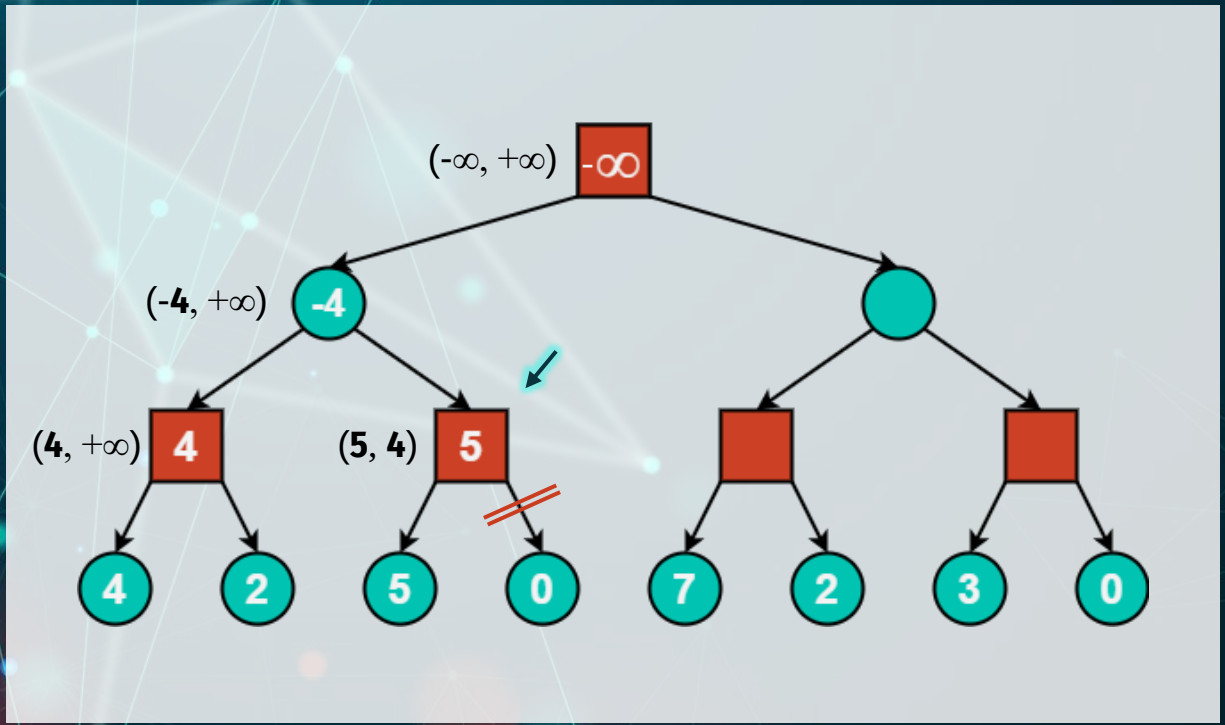
Zadatak 6 – Rešenje



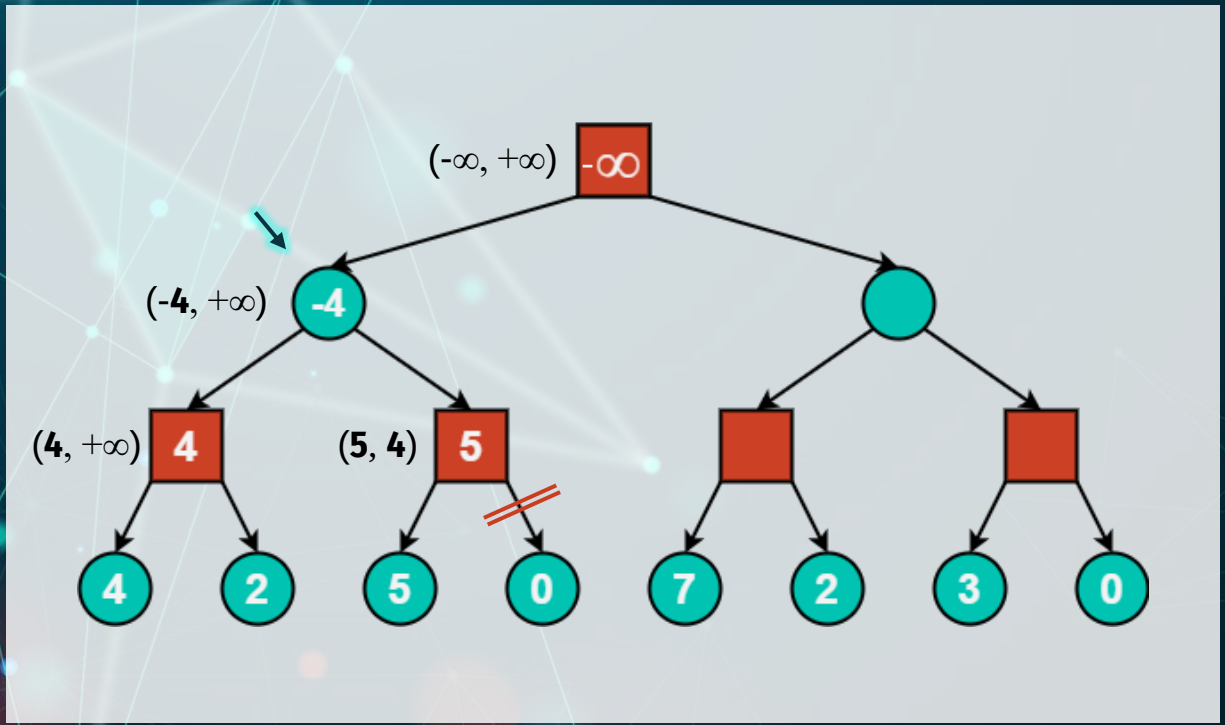
Zadatak 6 – Rešenje



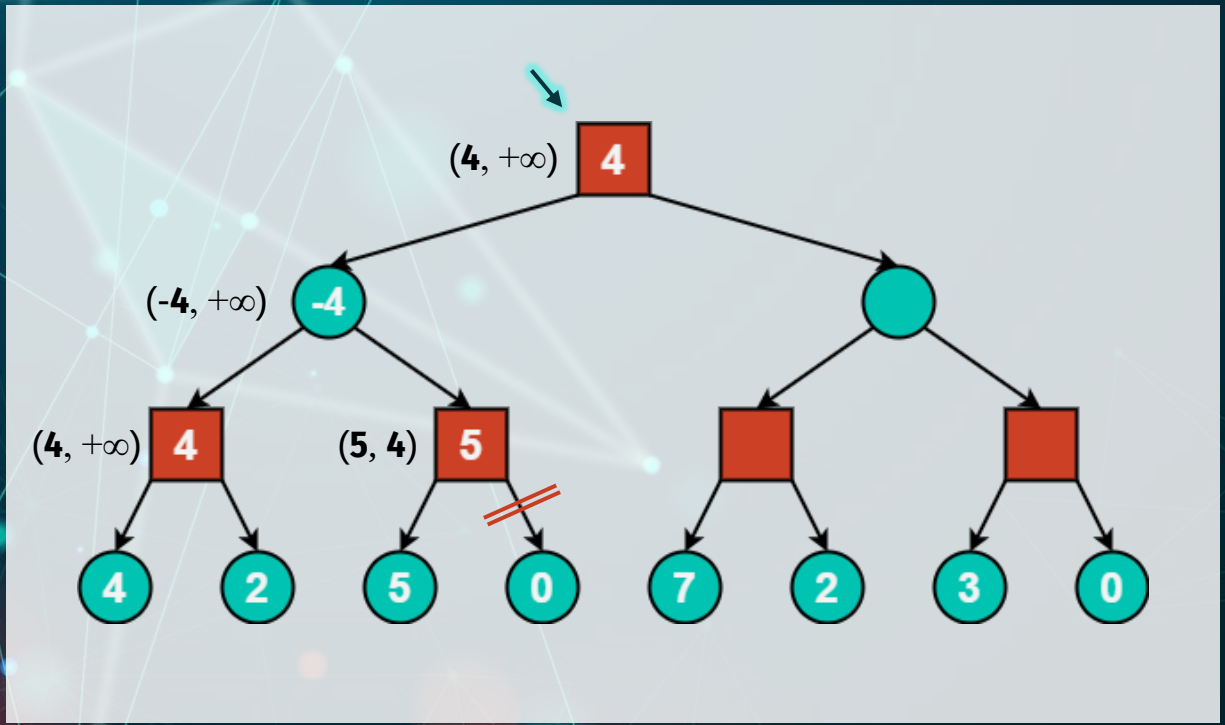
Zadatak 6 – Rešenje



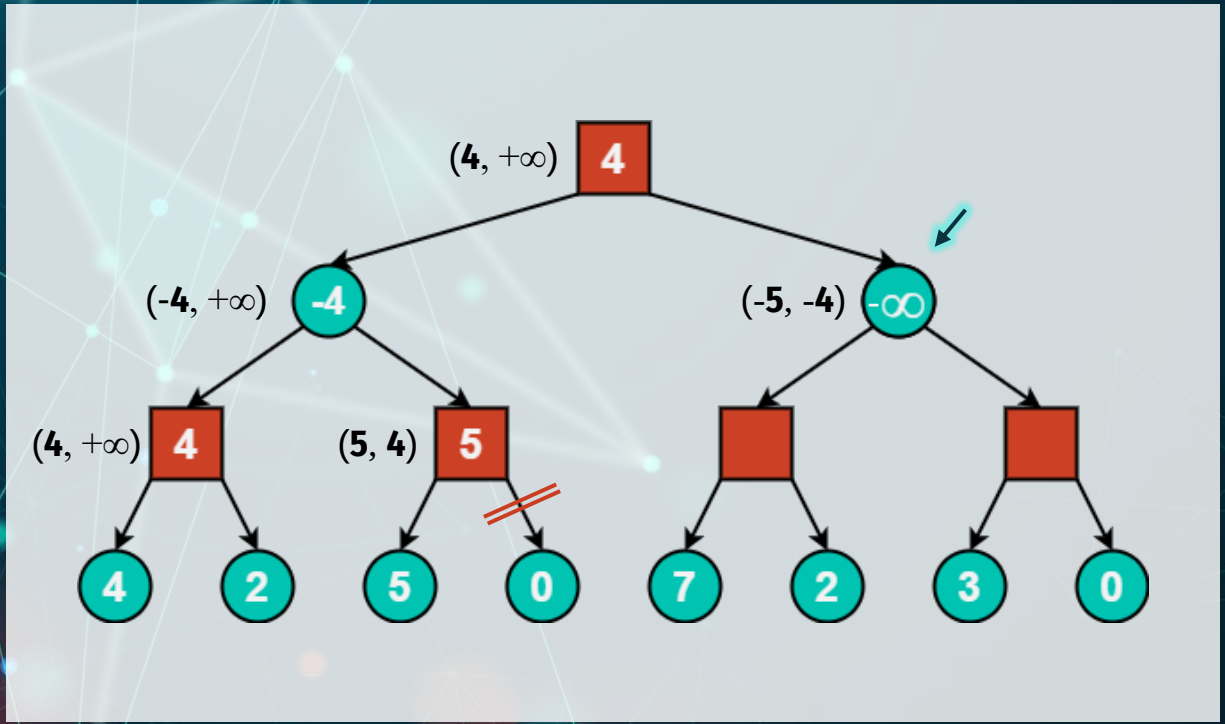
Zadatak 6 – Rešenje



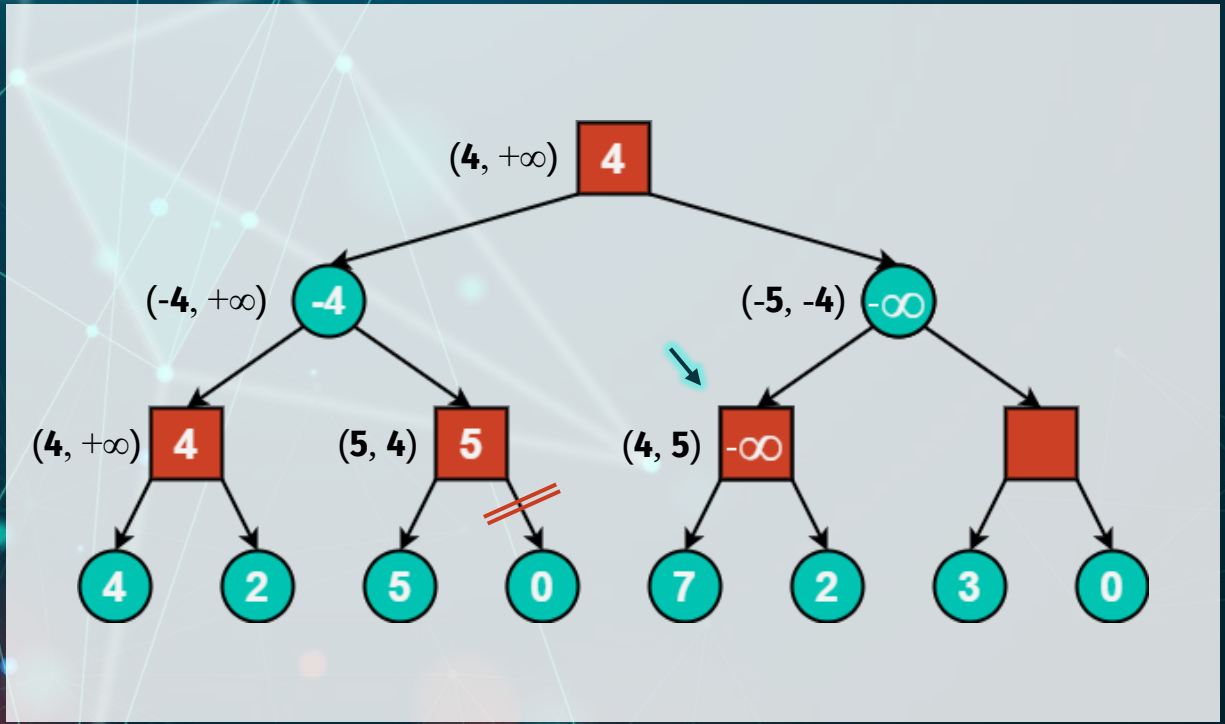
Zadatak 6 – Rešenje



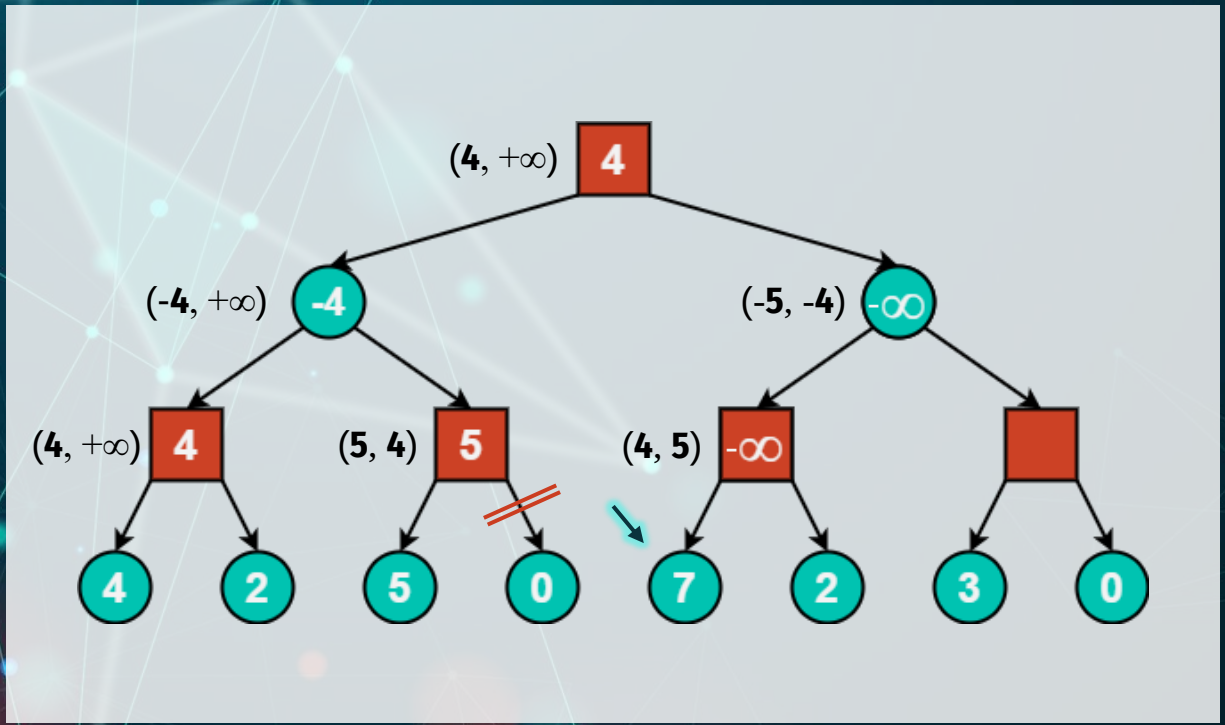
Zadatak 6 – Rešenje



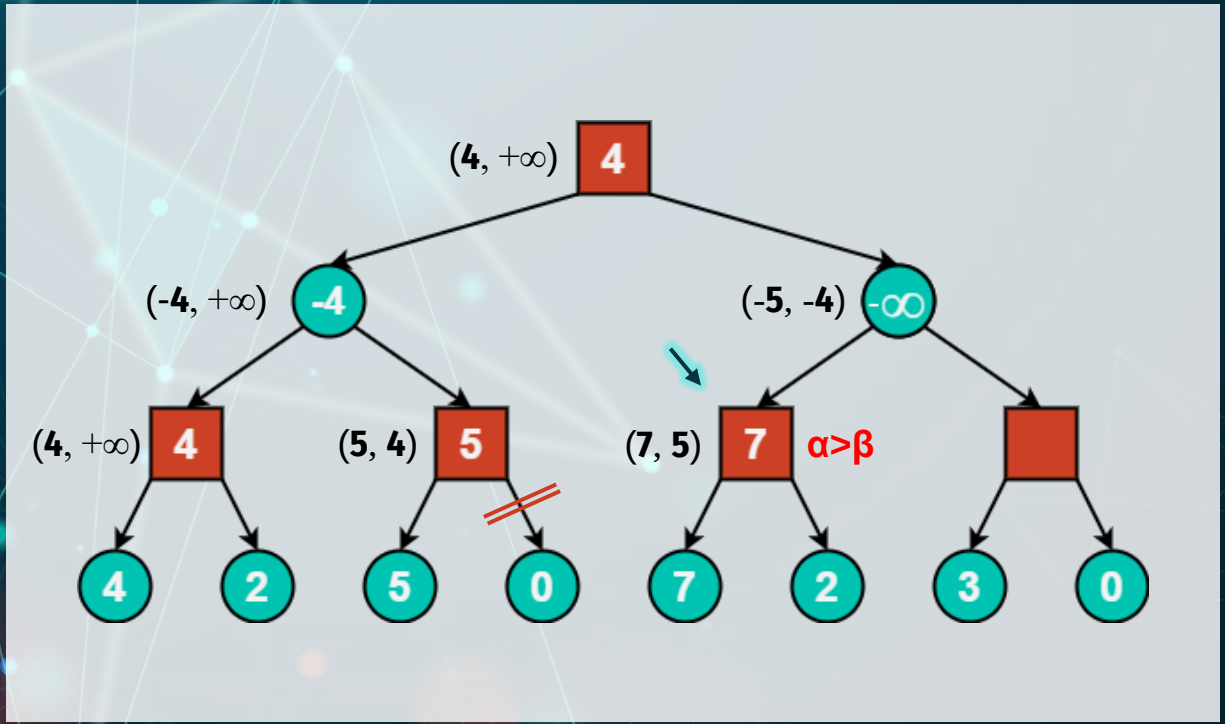
Zadatak 6 – Rešenje



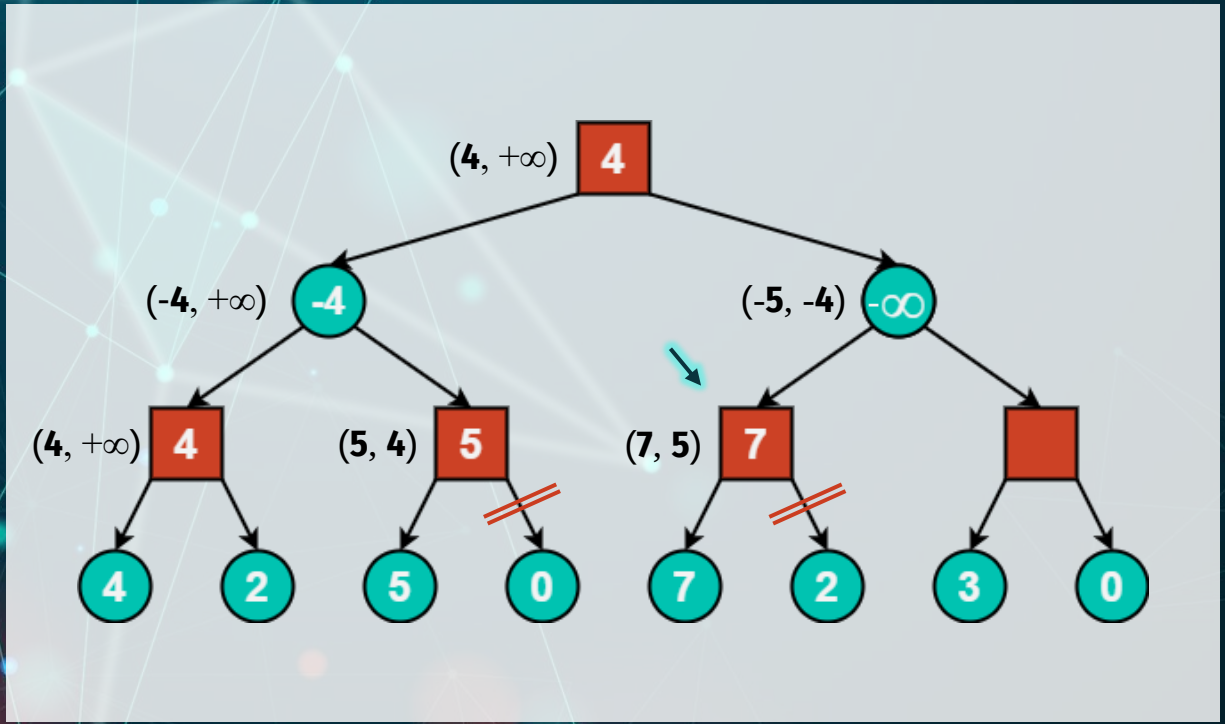
Zadatak 6 – Rešenje



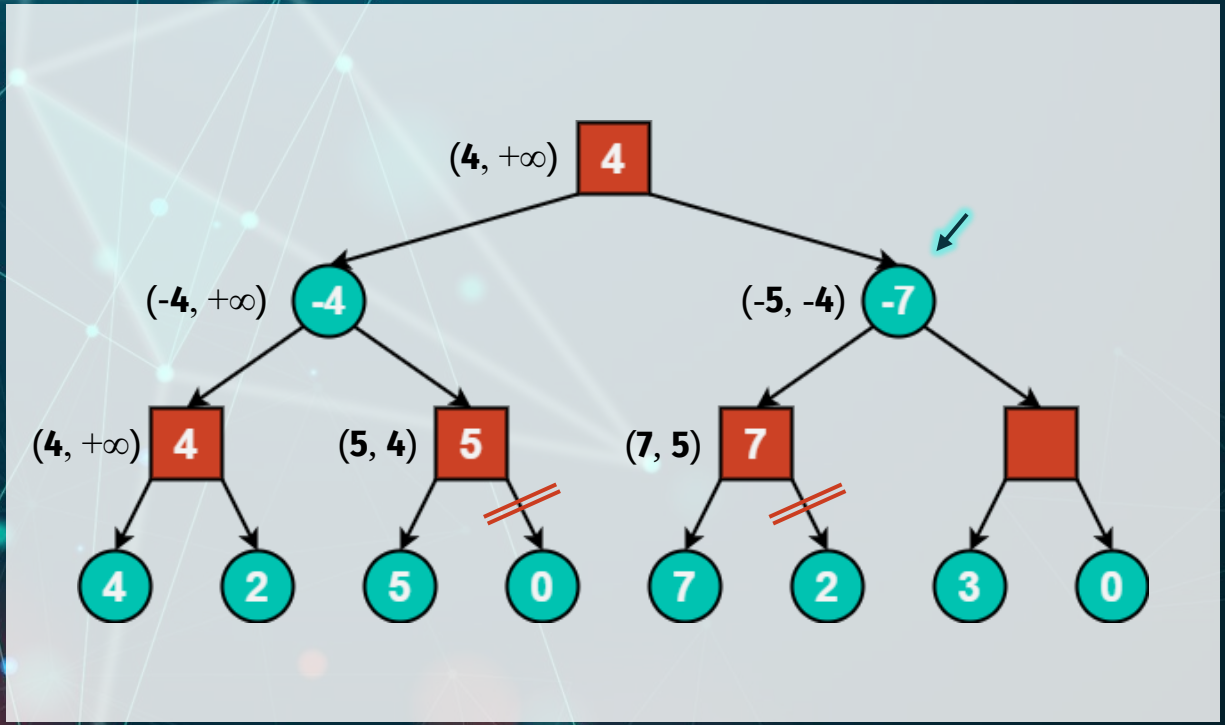
Zadatak 6 – Rešenje



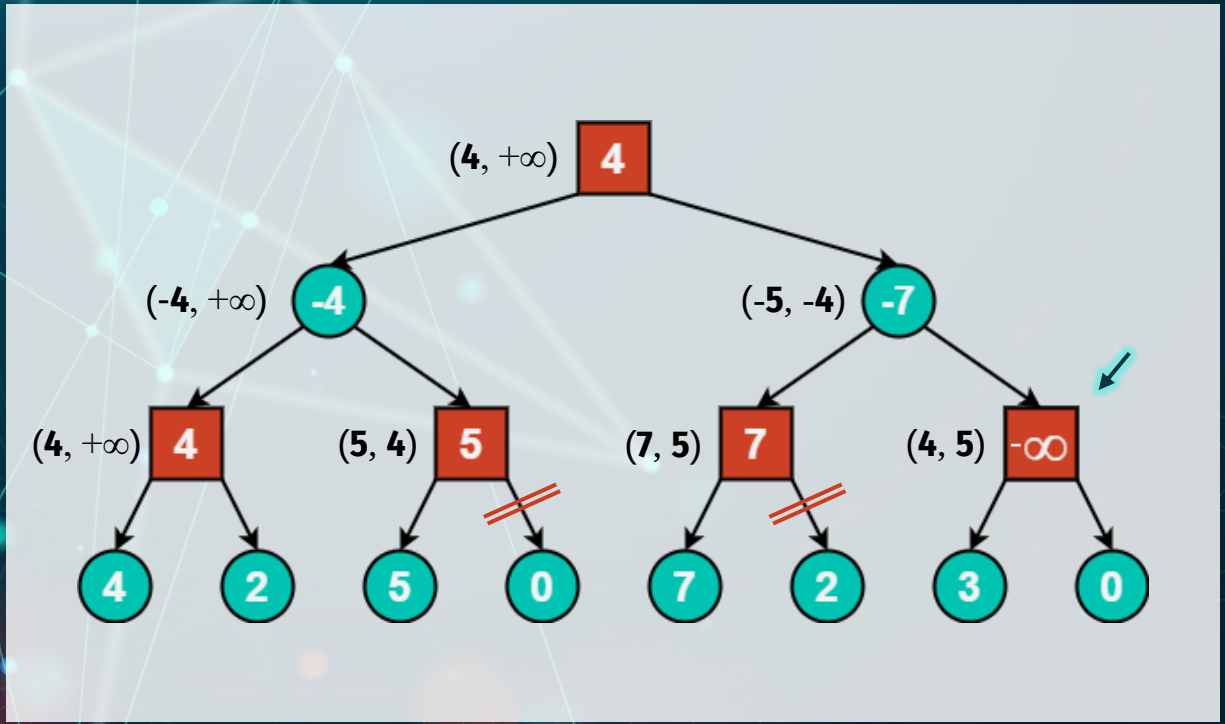
Zadatak 6 – Rešenje



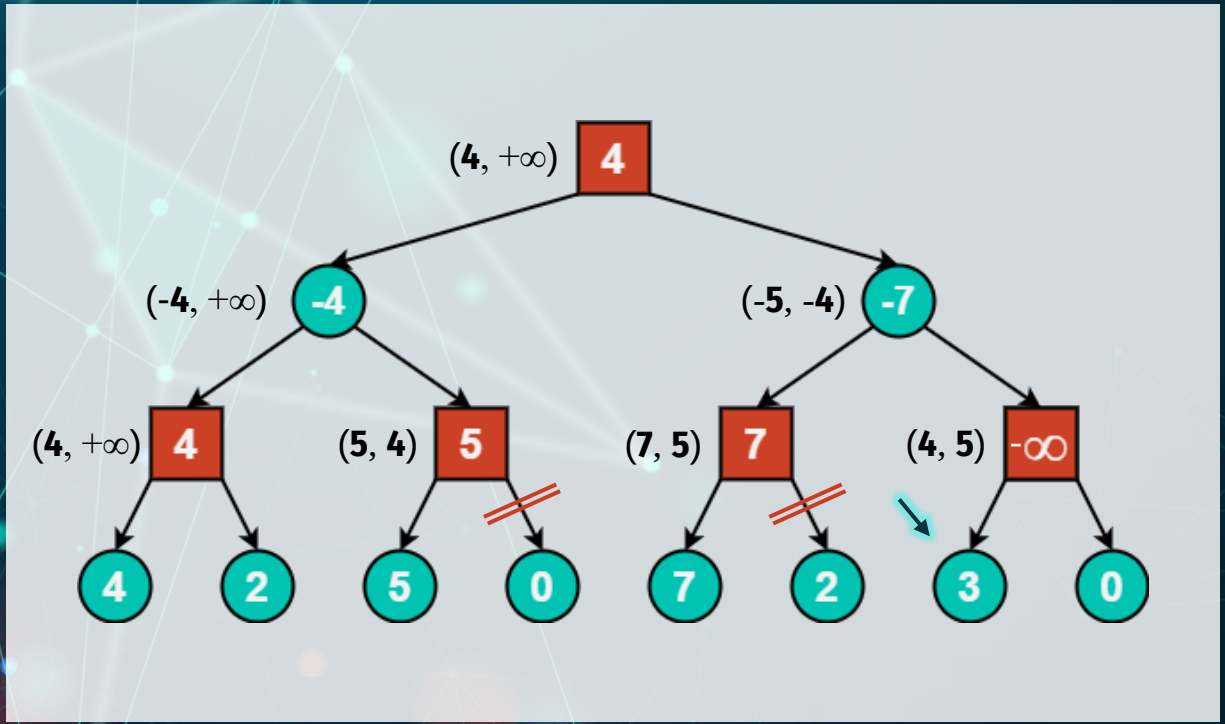
Zadatak 6 – Rešenje



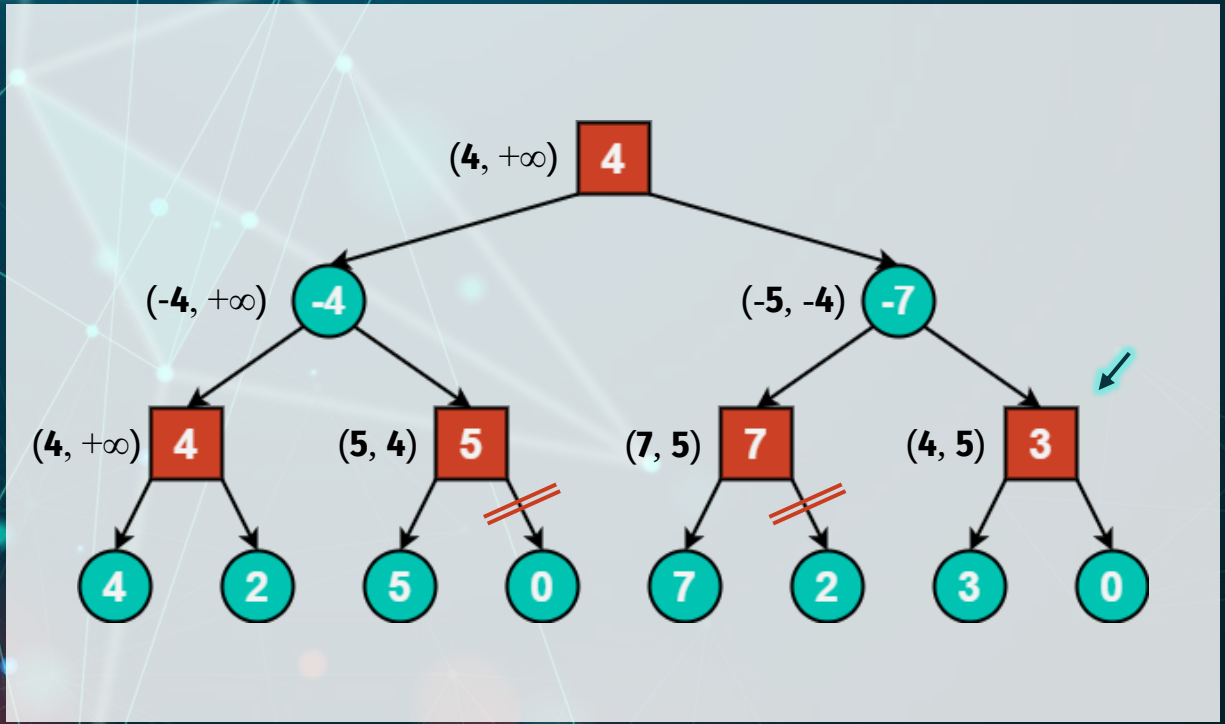
Zadatak 6 – Rešenje



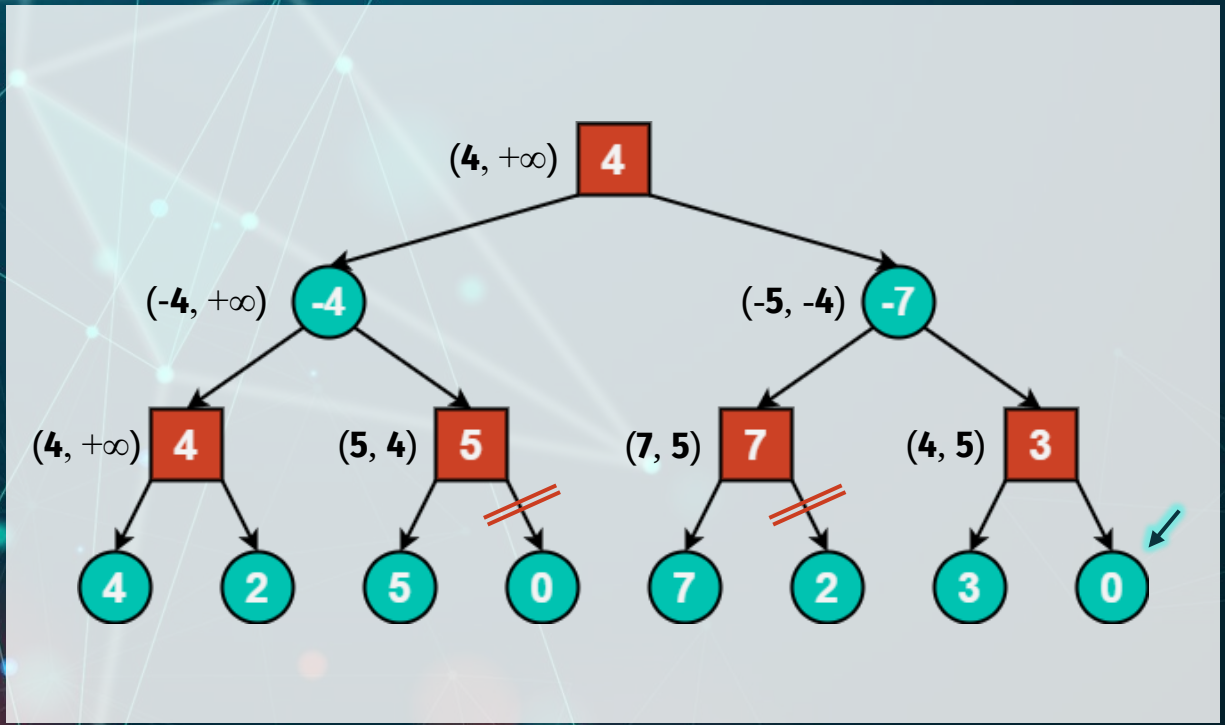
Zadatak 6 – Rešenje



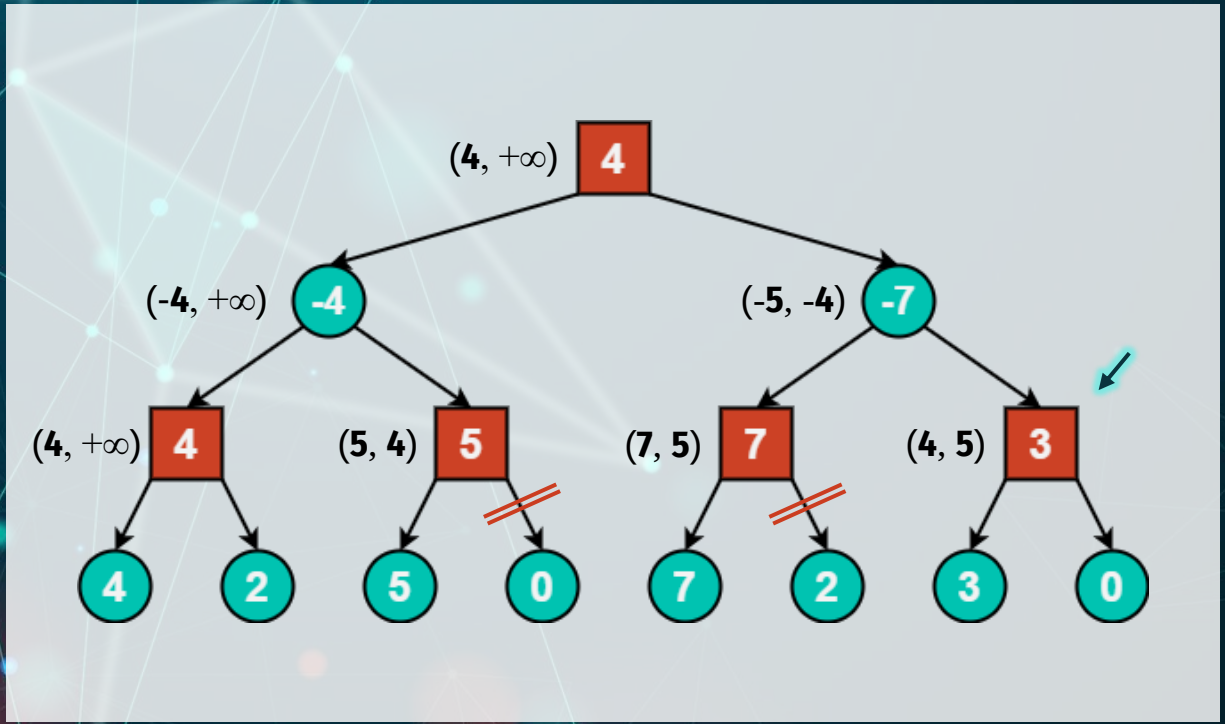
Zadatak 6 – Rešenje



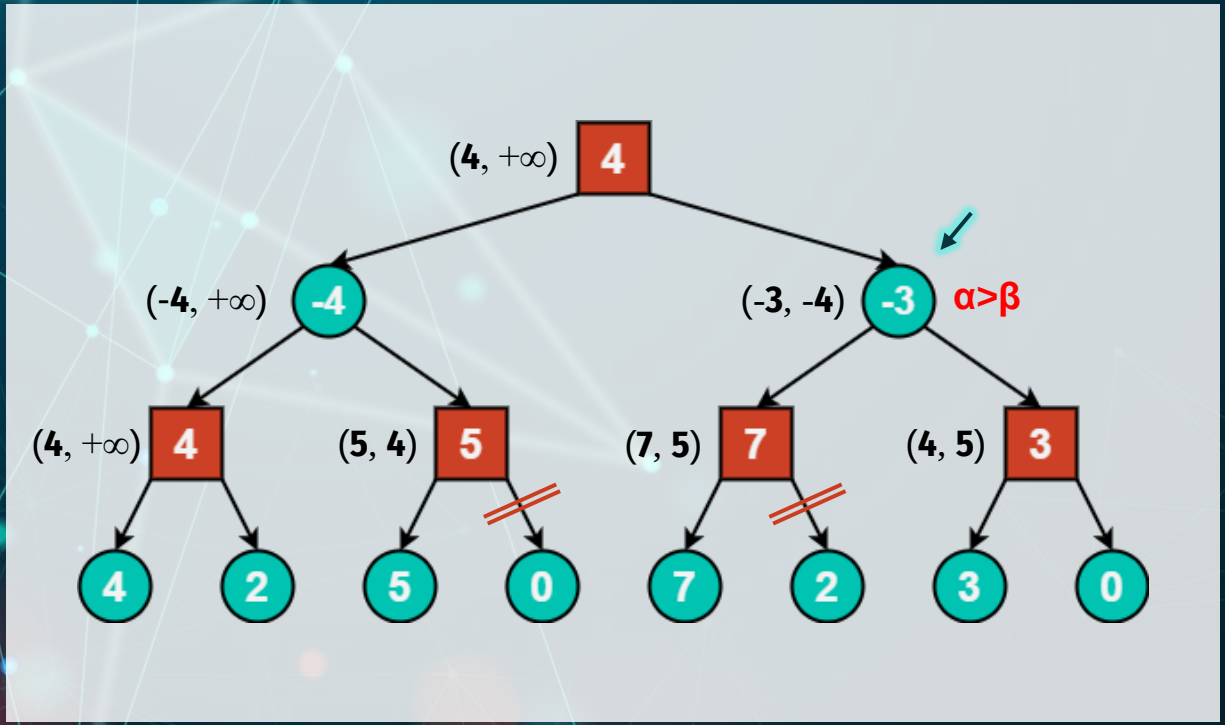
Zadatak 6 – Rešenje



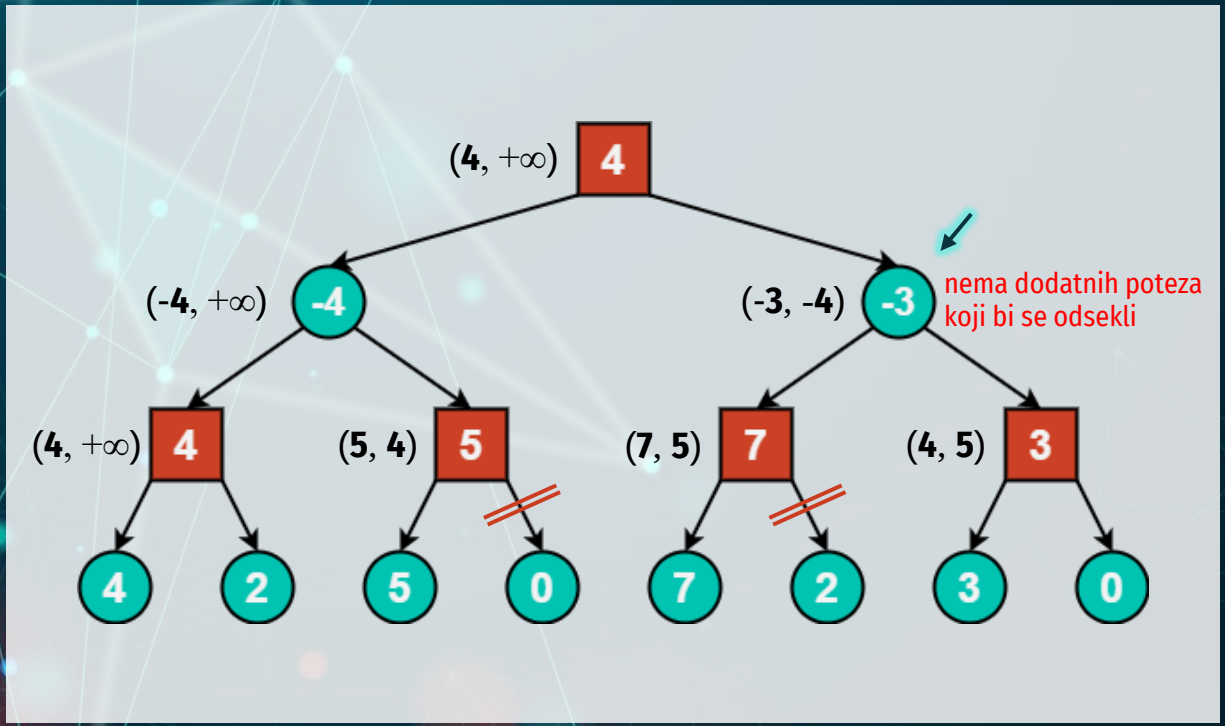
Zadatak 6 – Rešenje



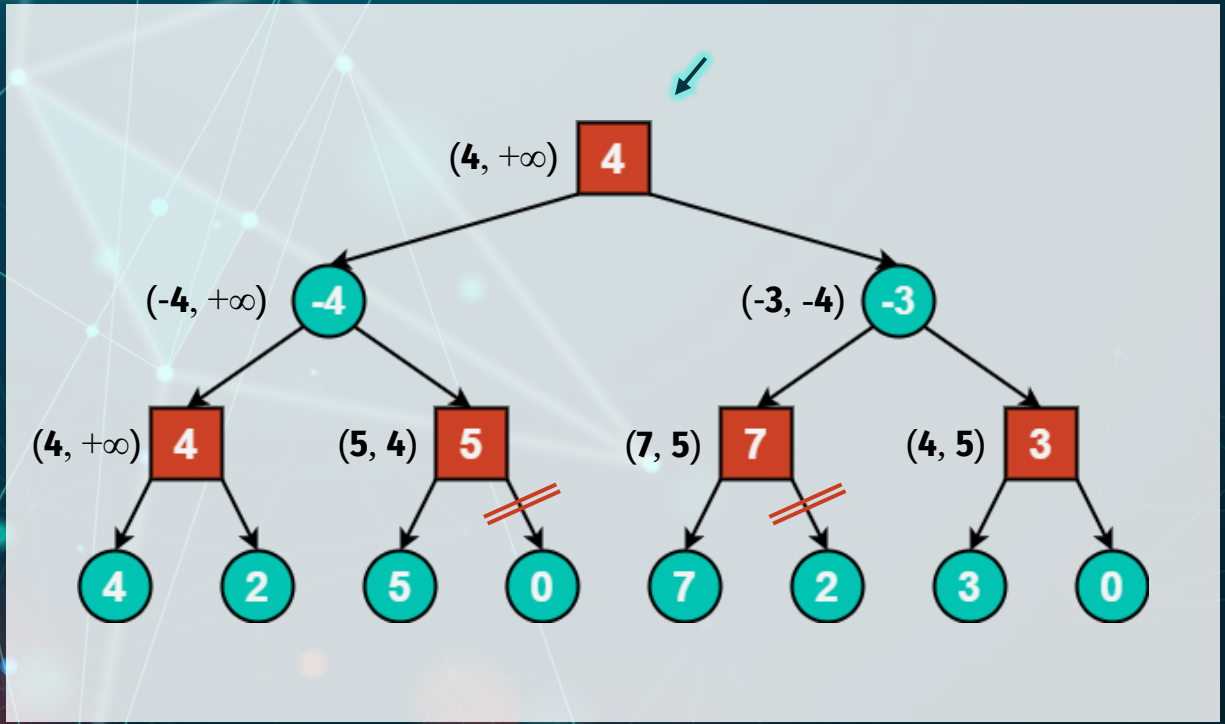
Zadatak 6 – Rešenje



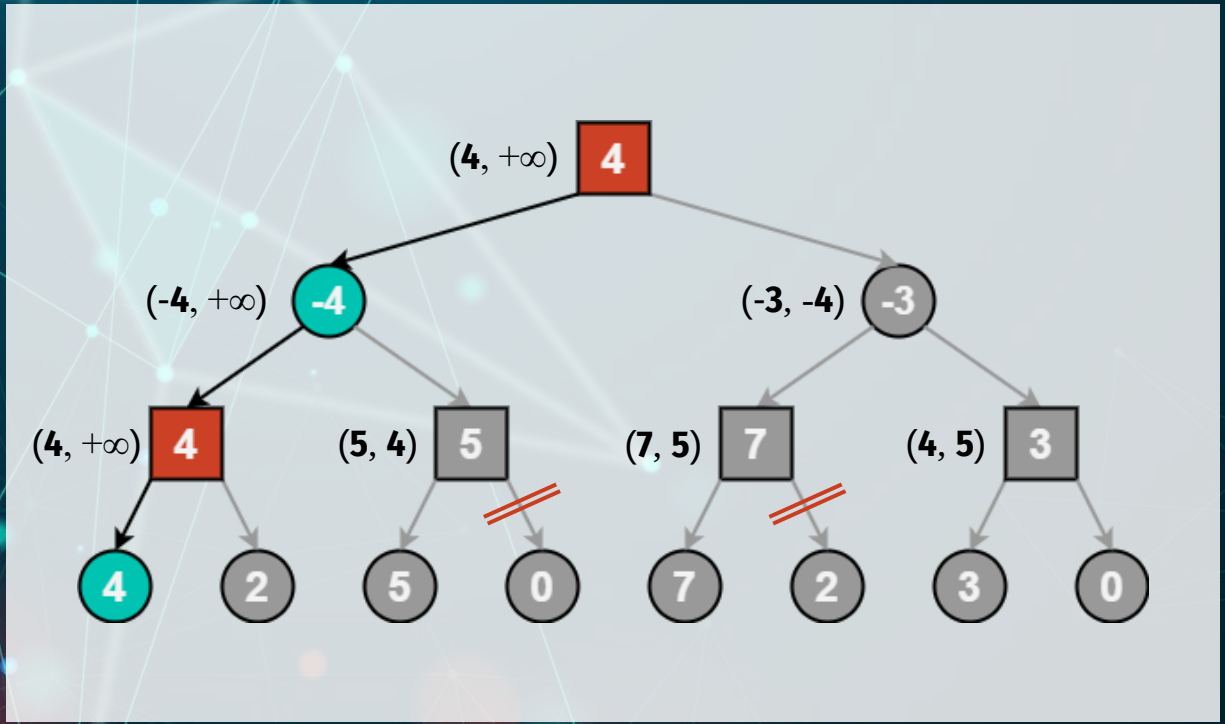
Zadatak 6 – Rešenje



Zadatak 6 – Rešenje

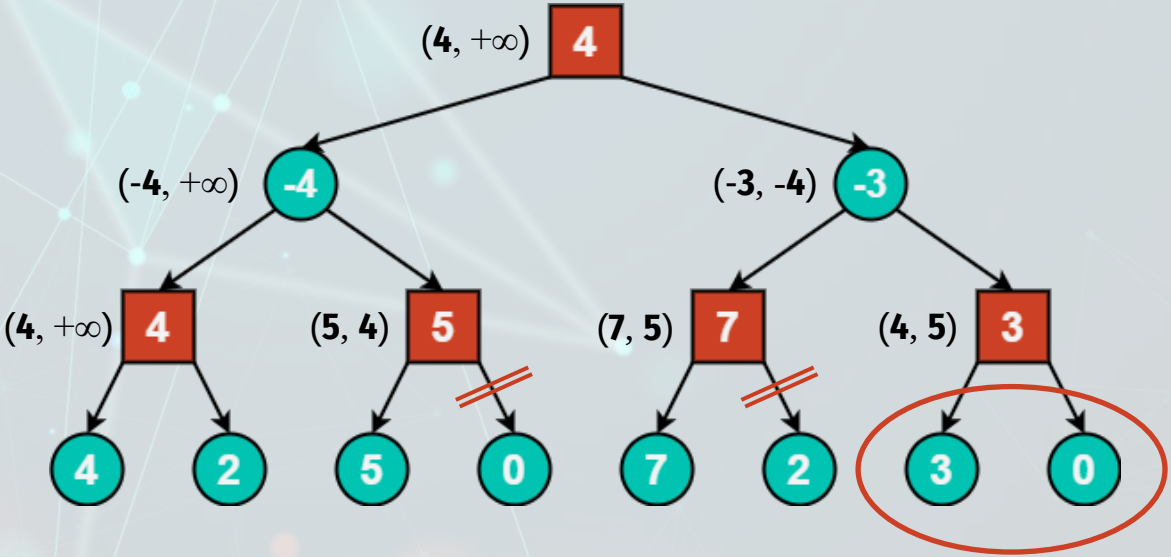


Zadatak 6 – Rešenje



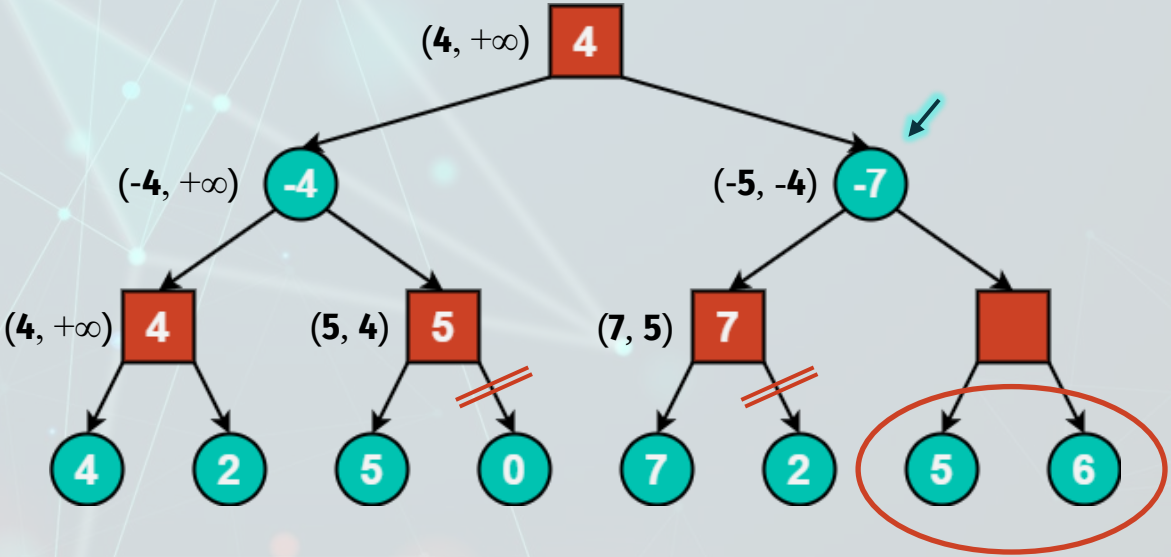
Zadatak 6 – Rešenje

Šta bi se desilo ukoliko bismo izmenili vrednosti označenog dela stabla u 5 i 6?

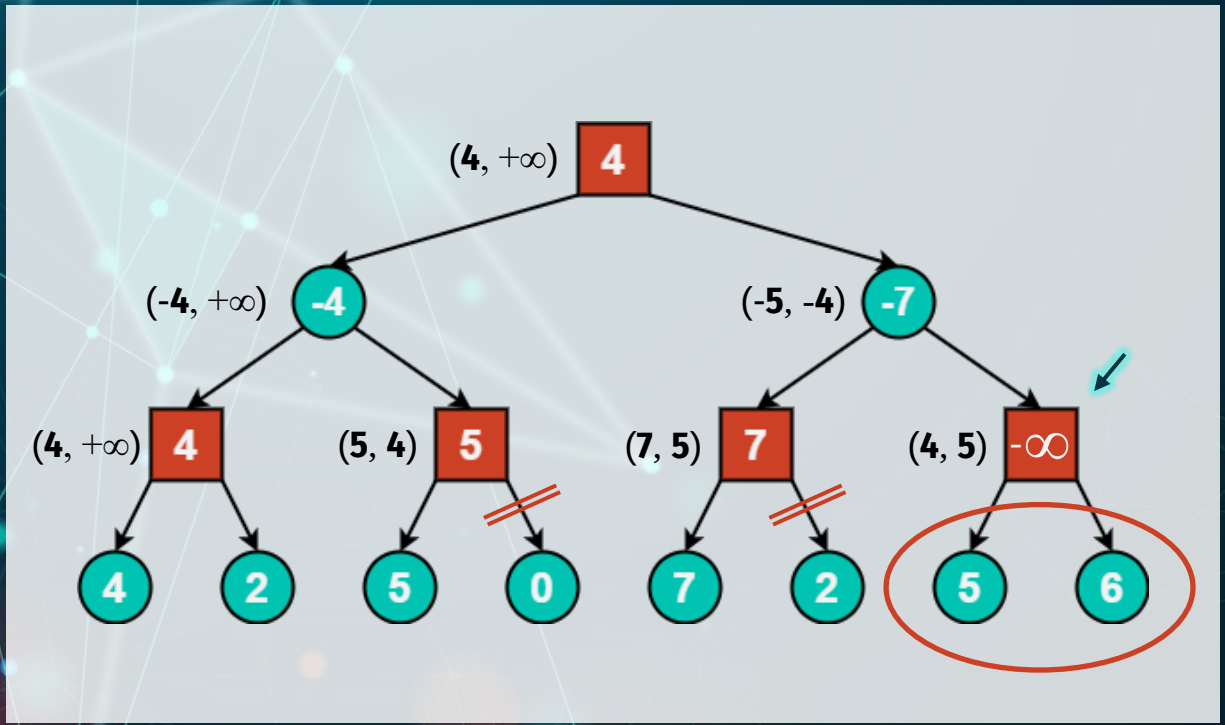


Zadatak 6 – Rešenje

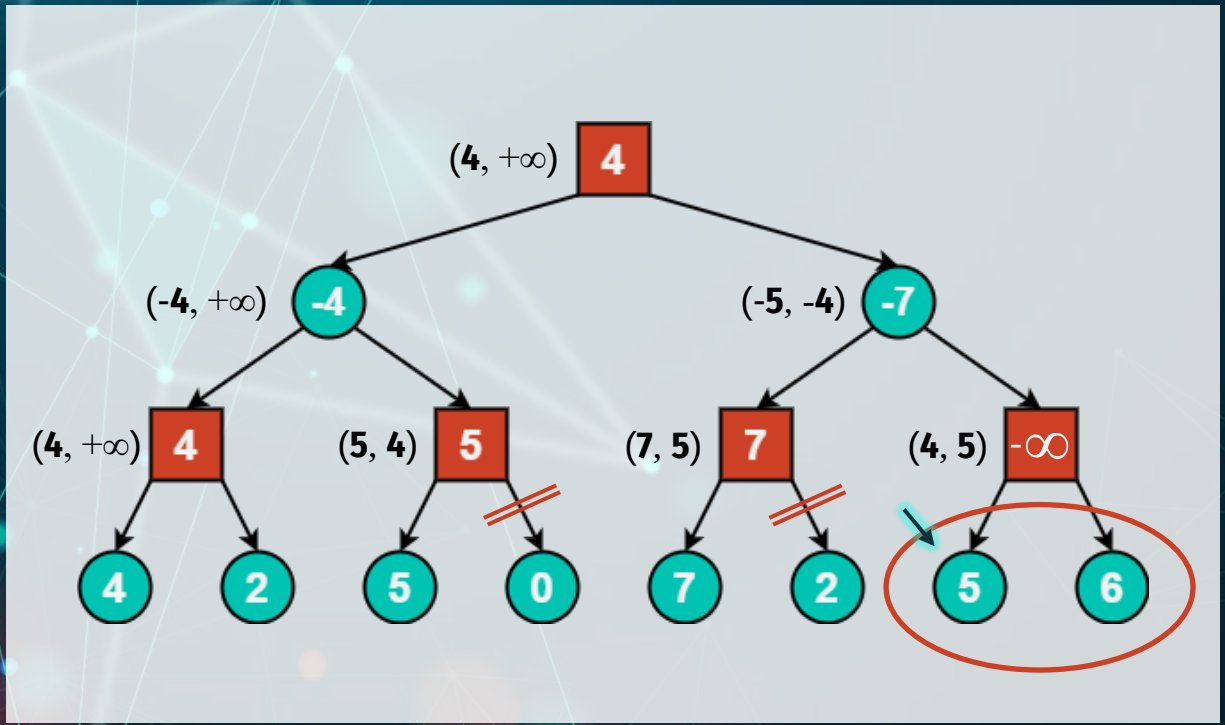
Deo stabla sa slike bi ostao isti.



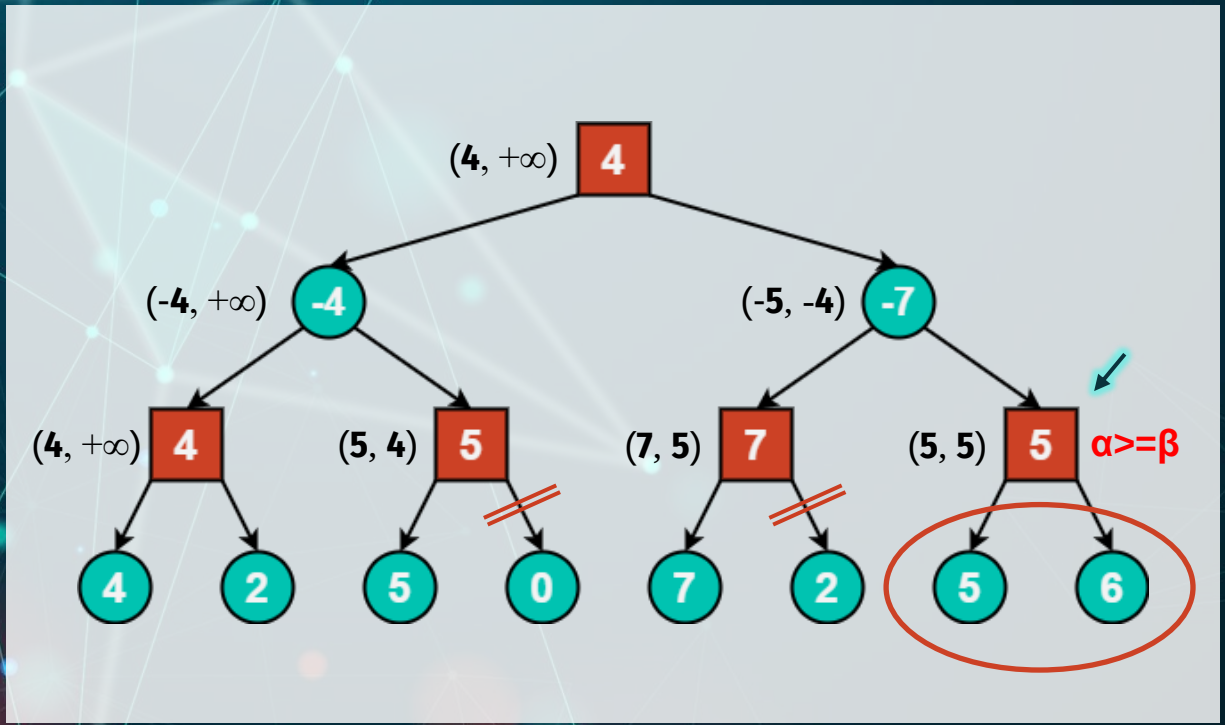
Zadatak 6 – Rešenje



Zadatak 6 – Rešenje

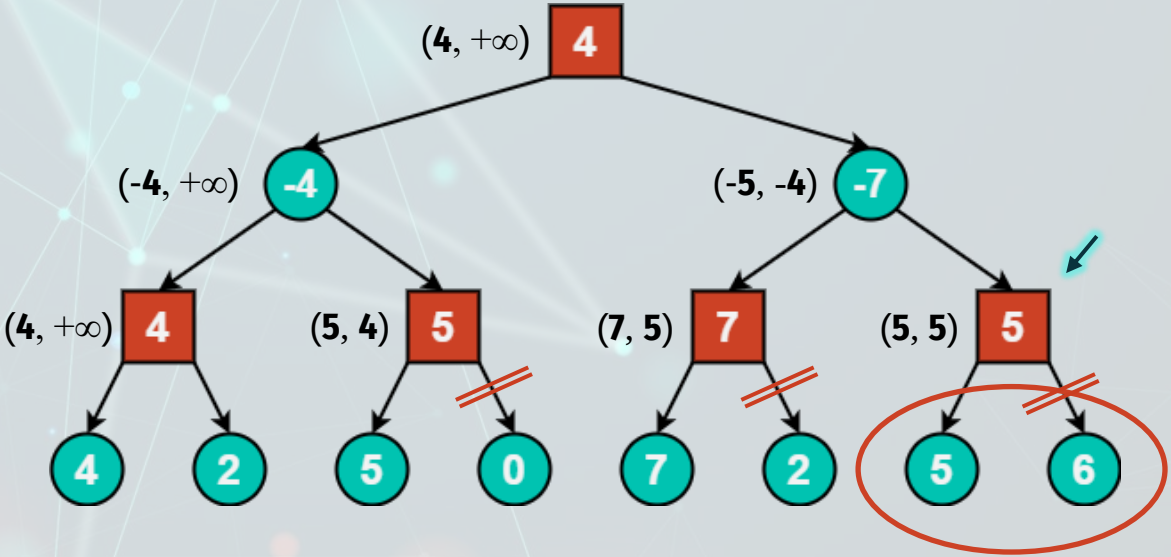


Zadatak 6 – Rešenje

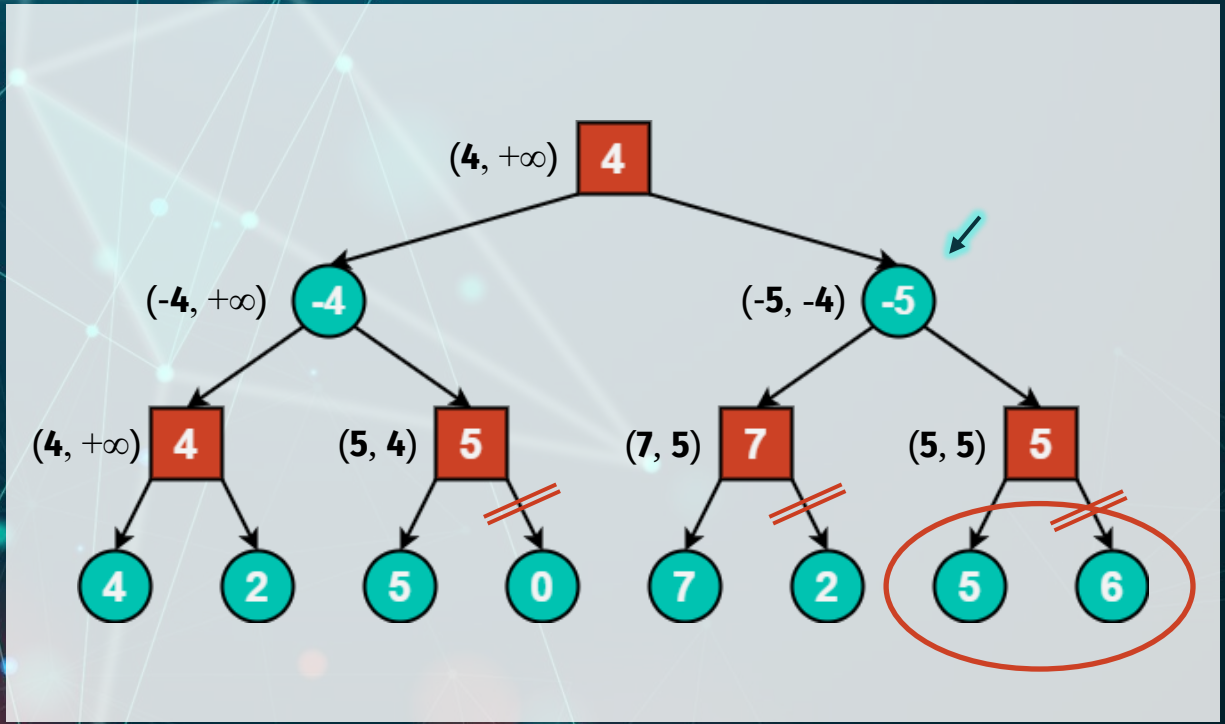


Zadatak 6 – Rešenje

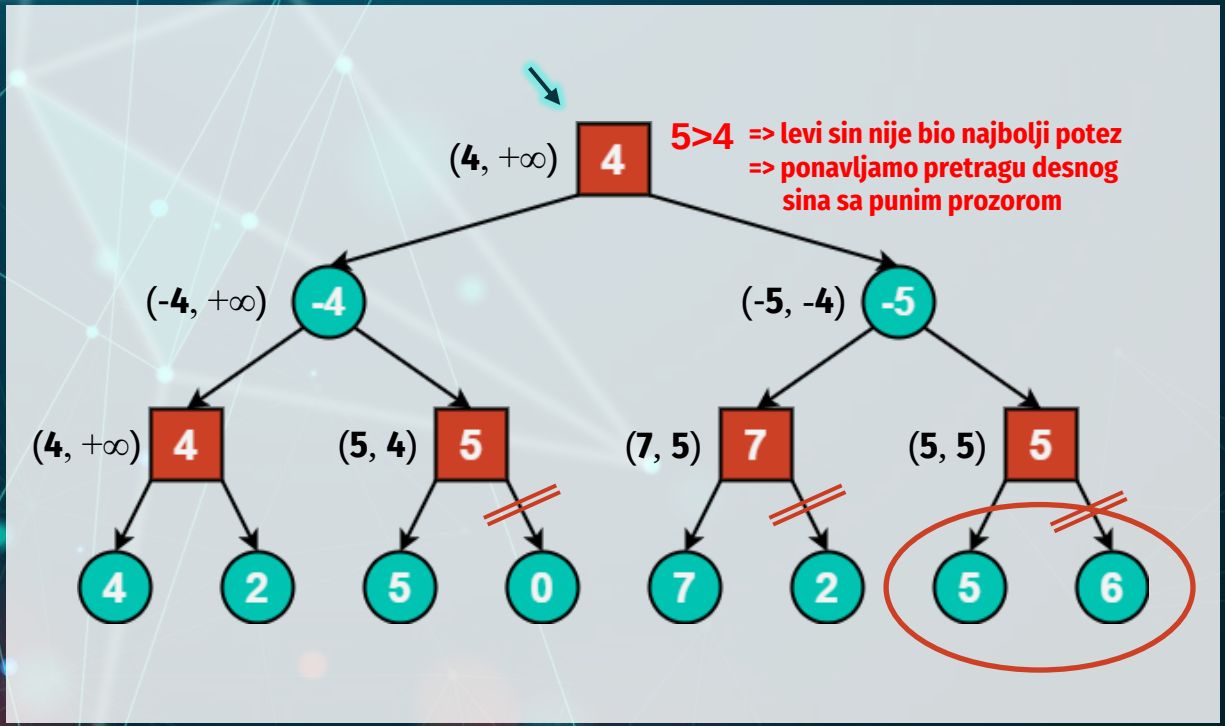
Da li smo potencijalno odbacili potez koji bi se na kraju odigrao?



Zadatak 6 – Rešenje

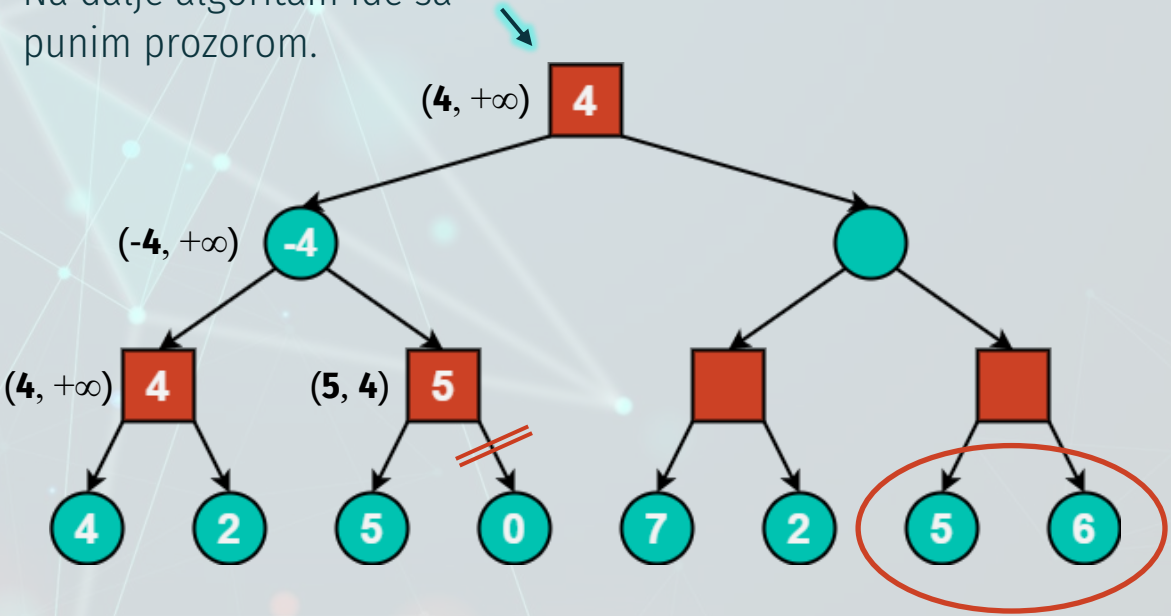


Zadatak 6 – Rešenje



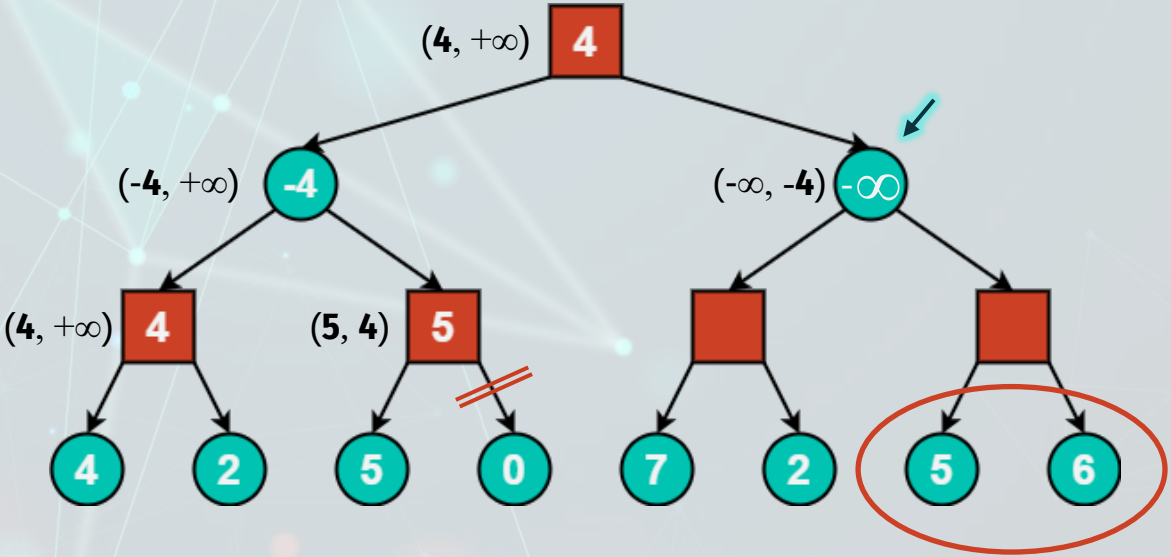
Zadatak 6 – Rešenje

Na dalje algoritam ide sa punim prozorom.

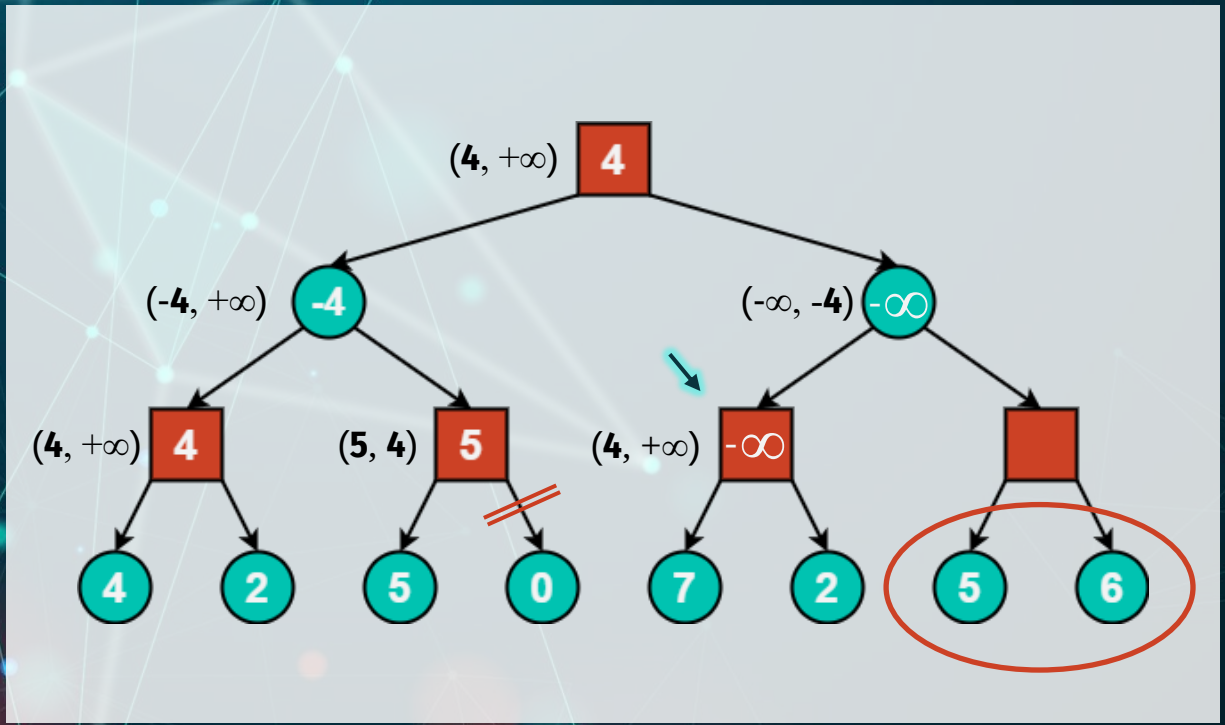


Zadatak 6 – Rešenje

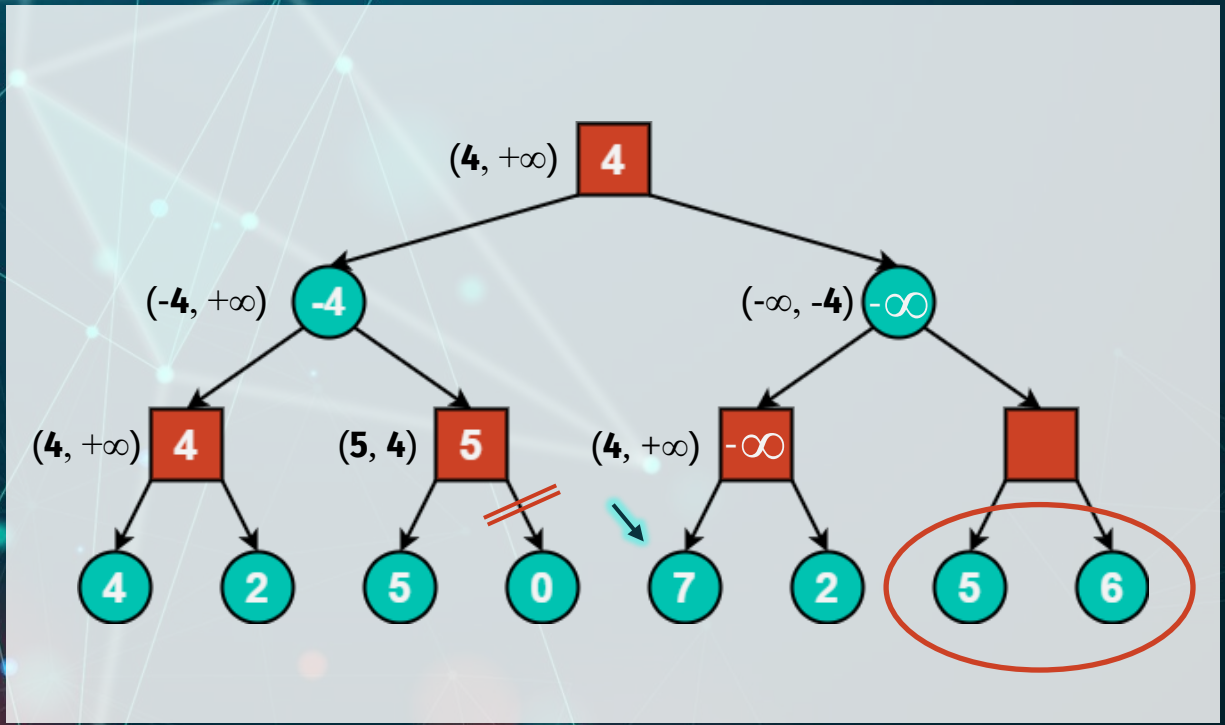
Na dalje algoritam ide sa punim prozorom.



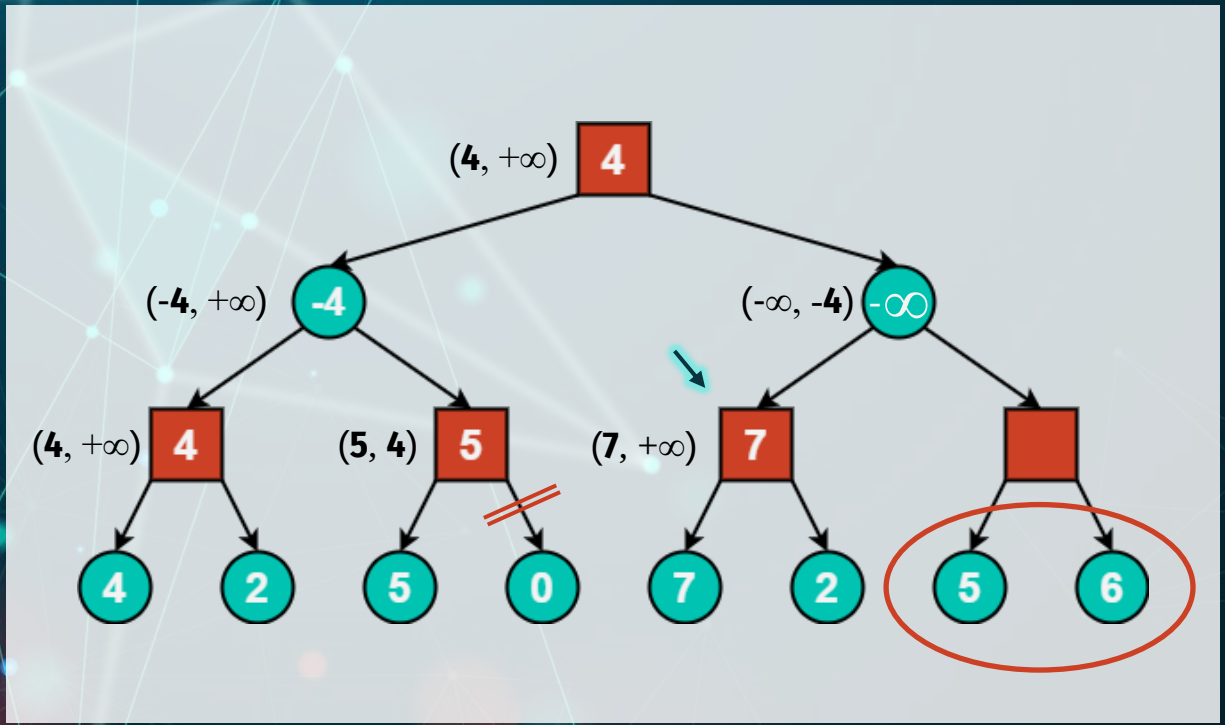
Zadatak 6 – Rešenje



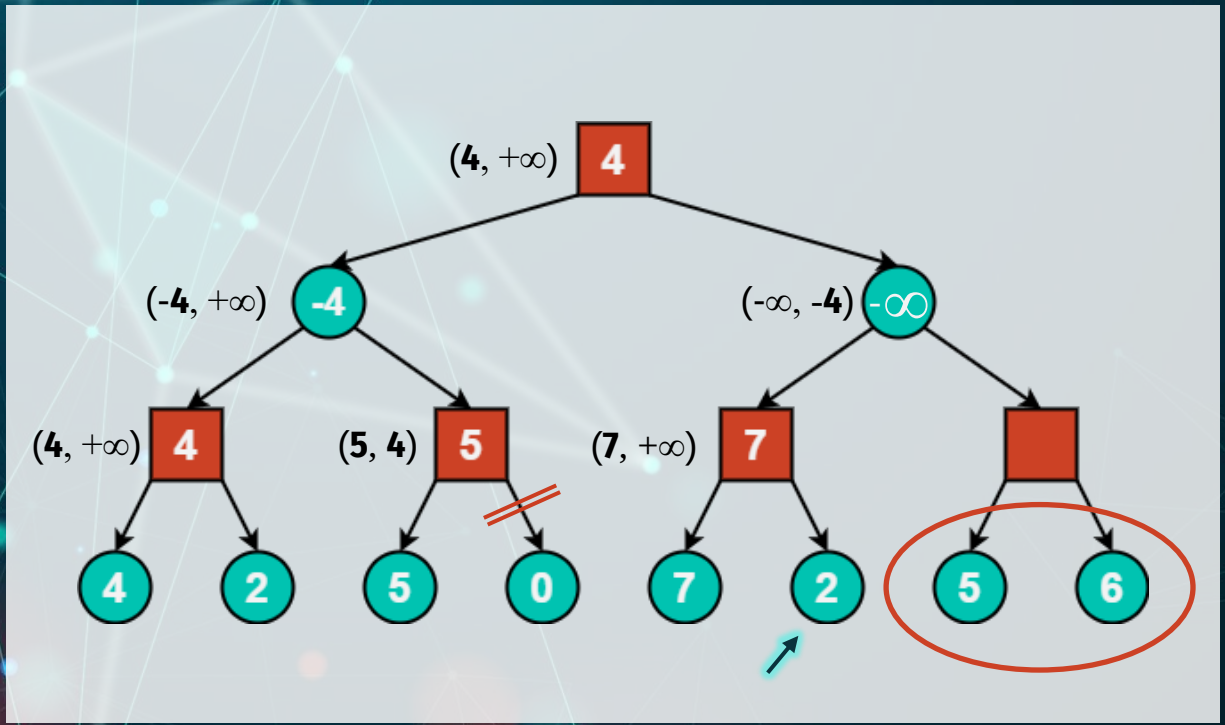
Zadatak 6 – Rešenje



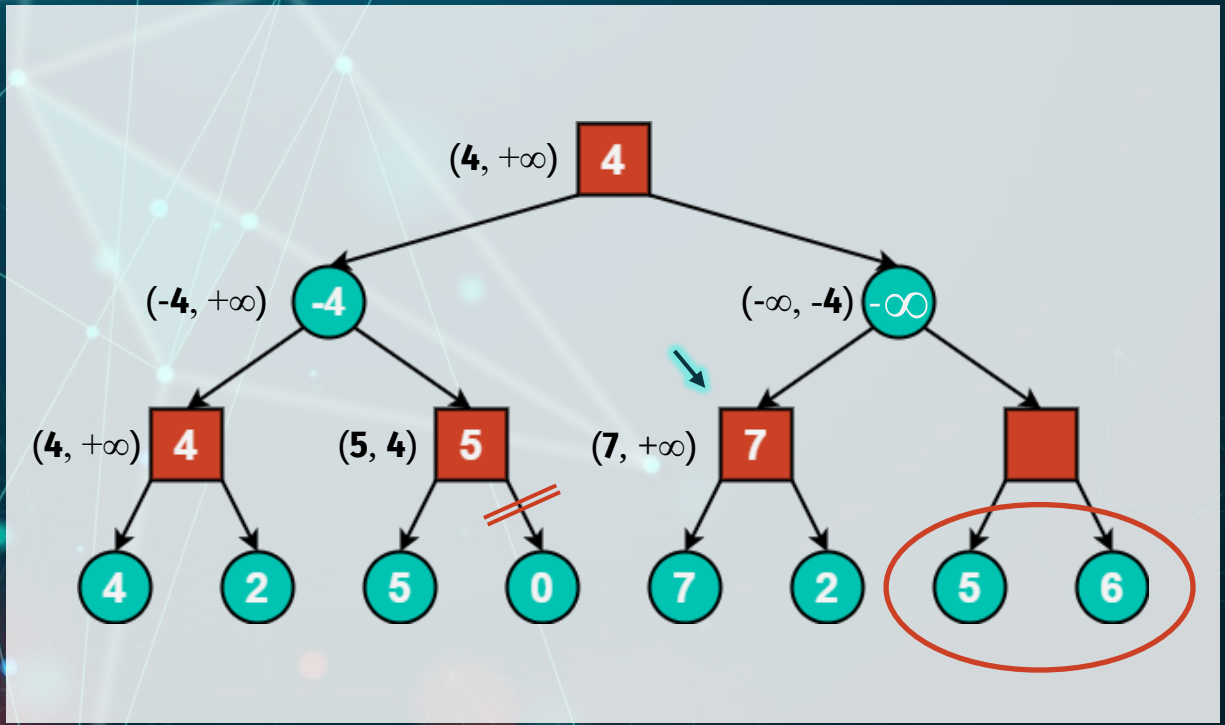
Zadatak 6 – Rešenje



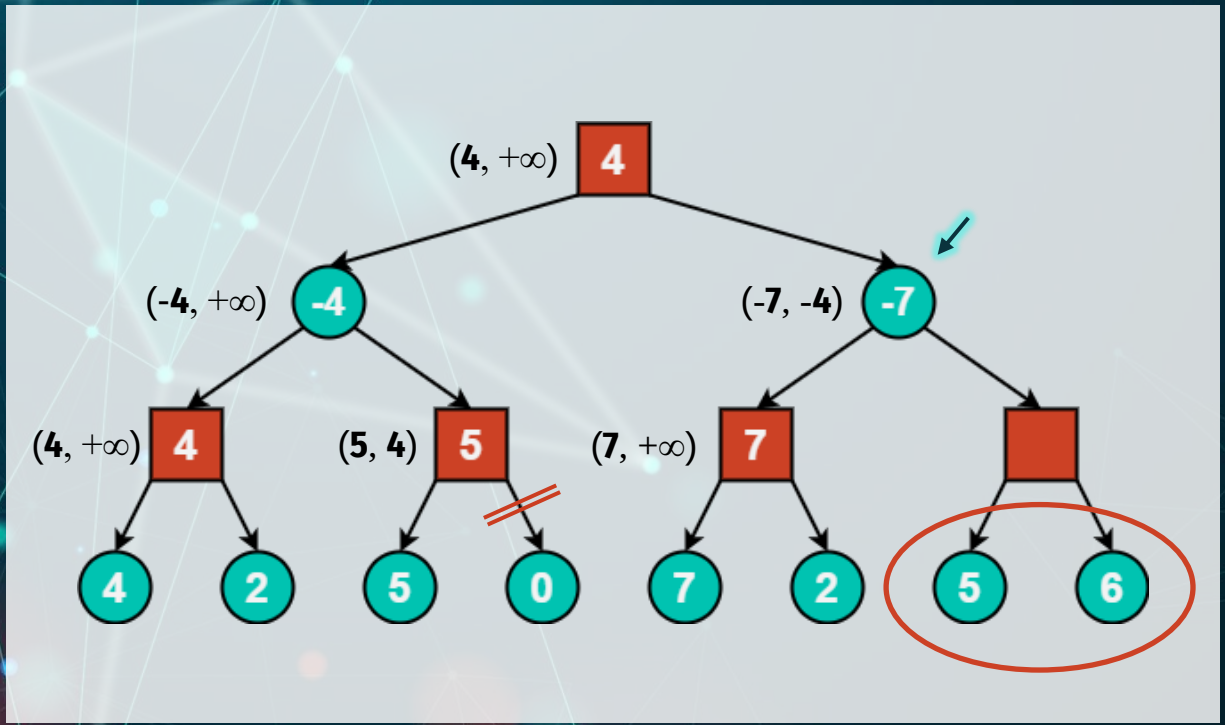
Zadatak 6 – Rešenje



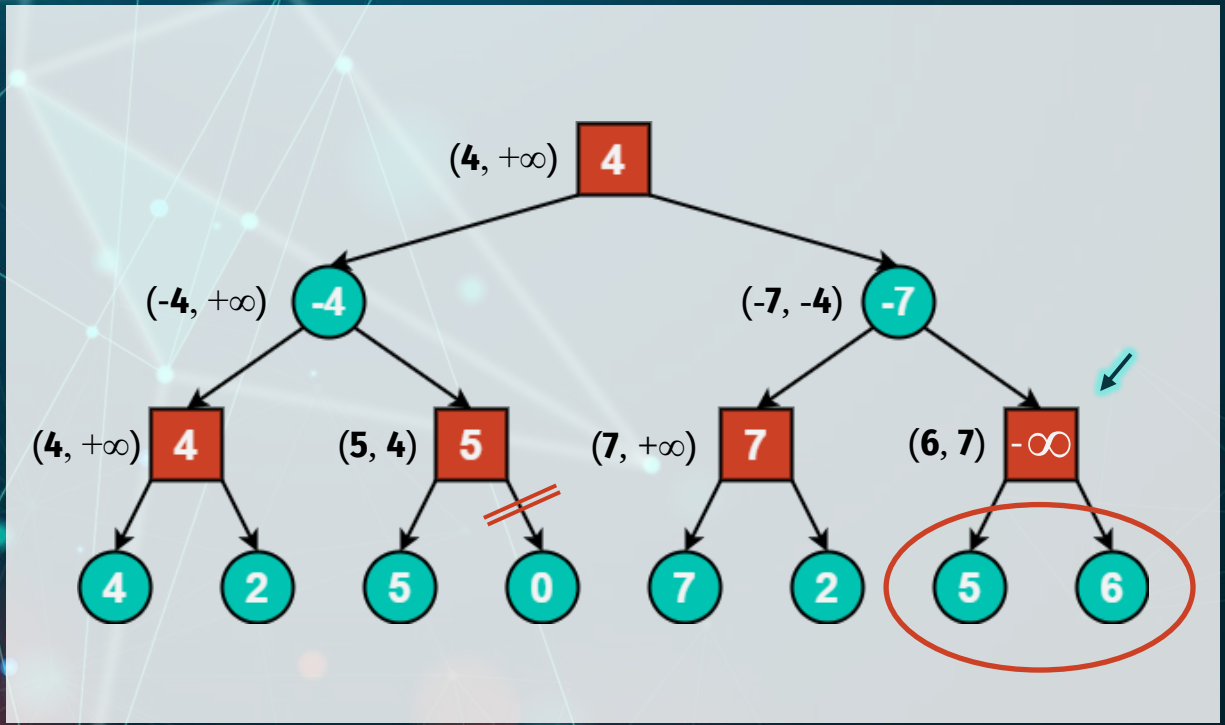
Zadatak 6 – Rešenje



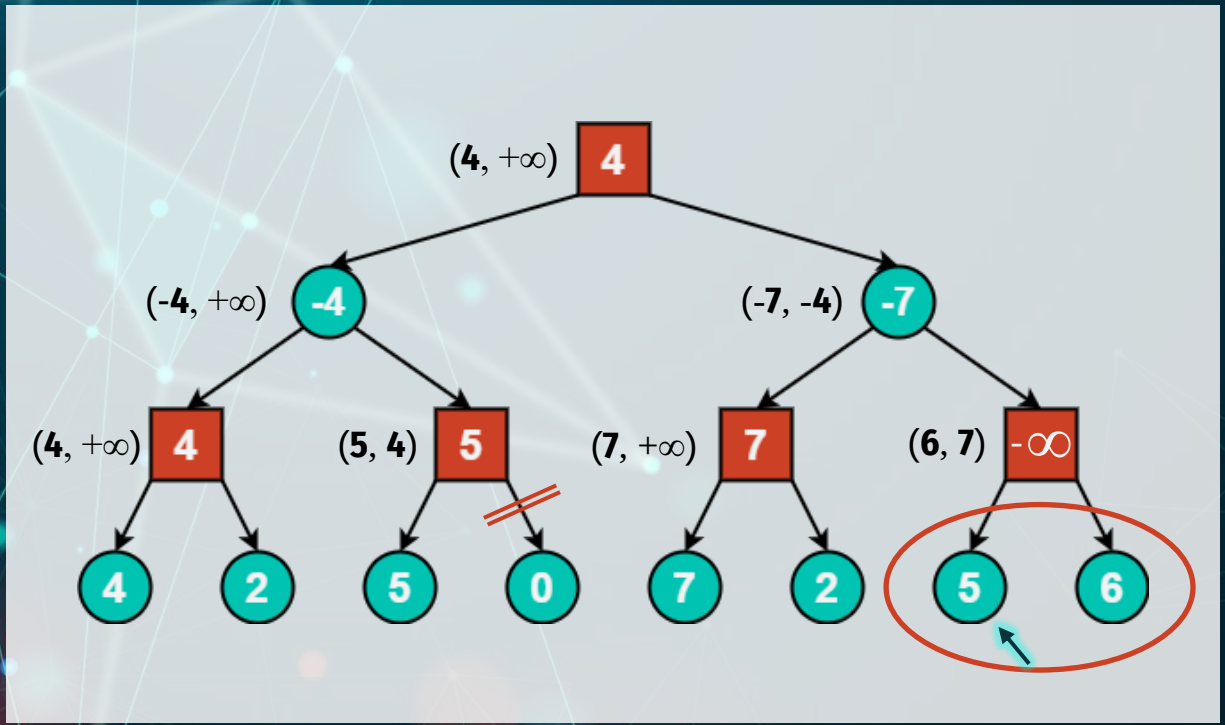
Zadatak 6 – Rešenje



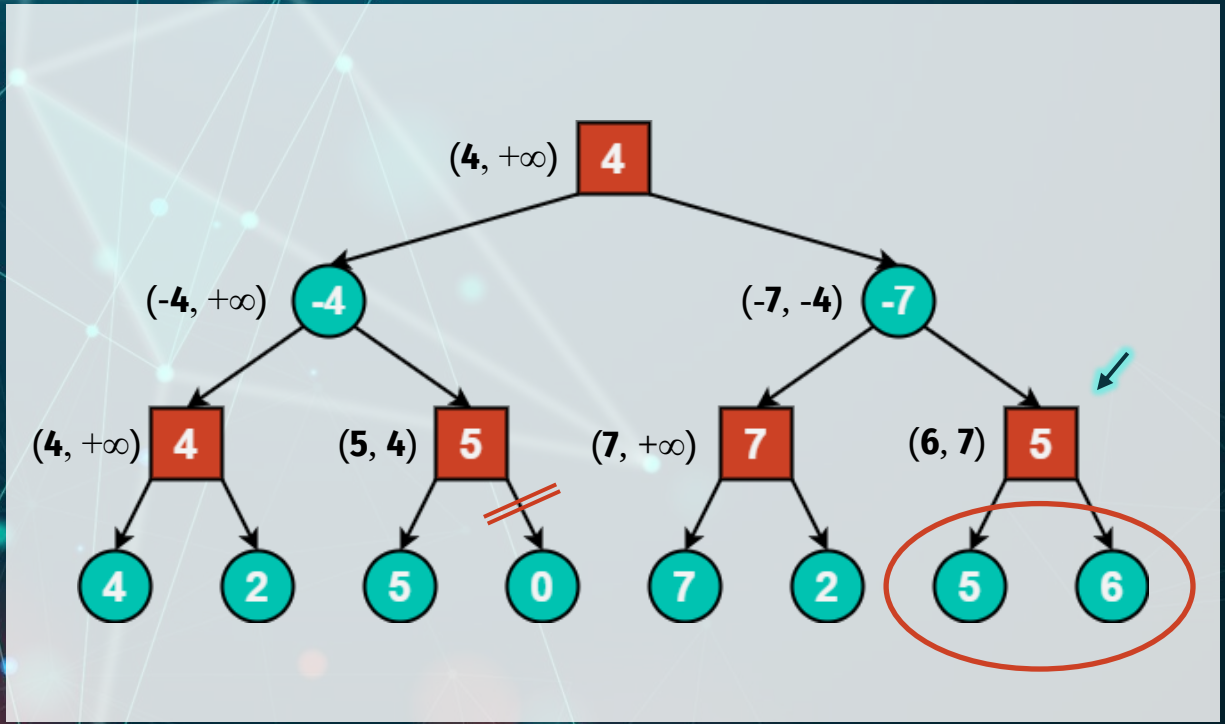
Zadatak 6 – Rešenje



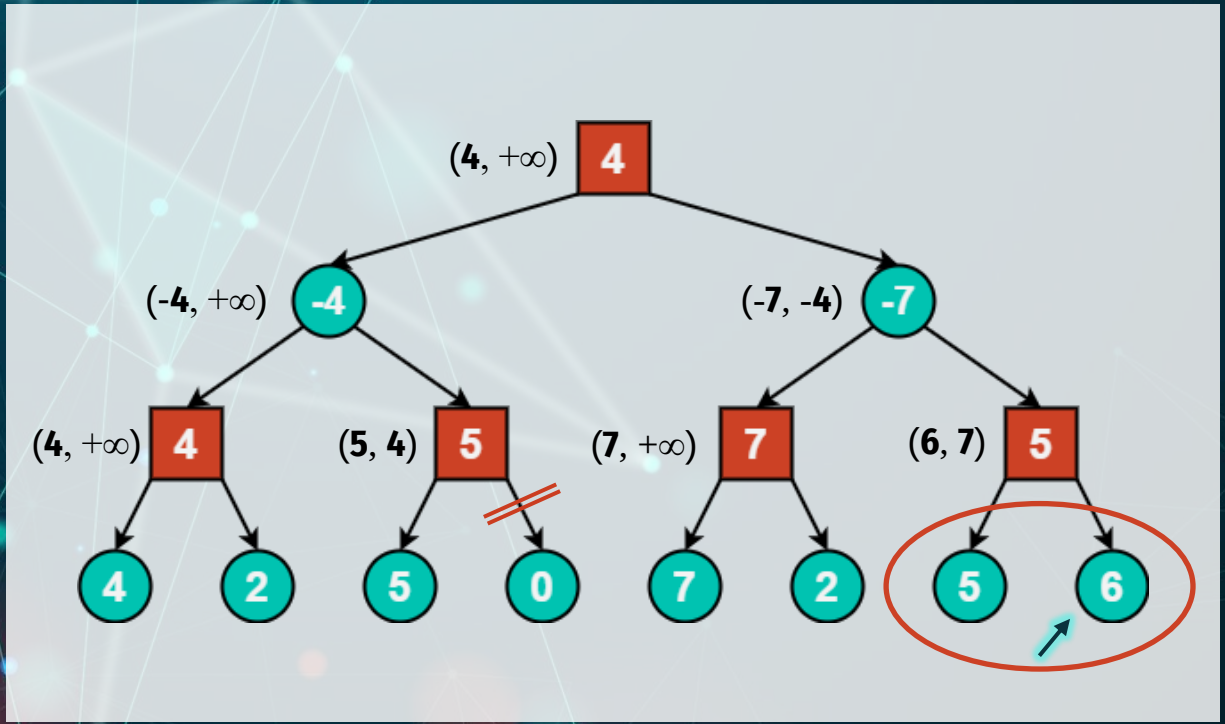
Zadatak 6 – Rešenje



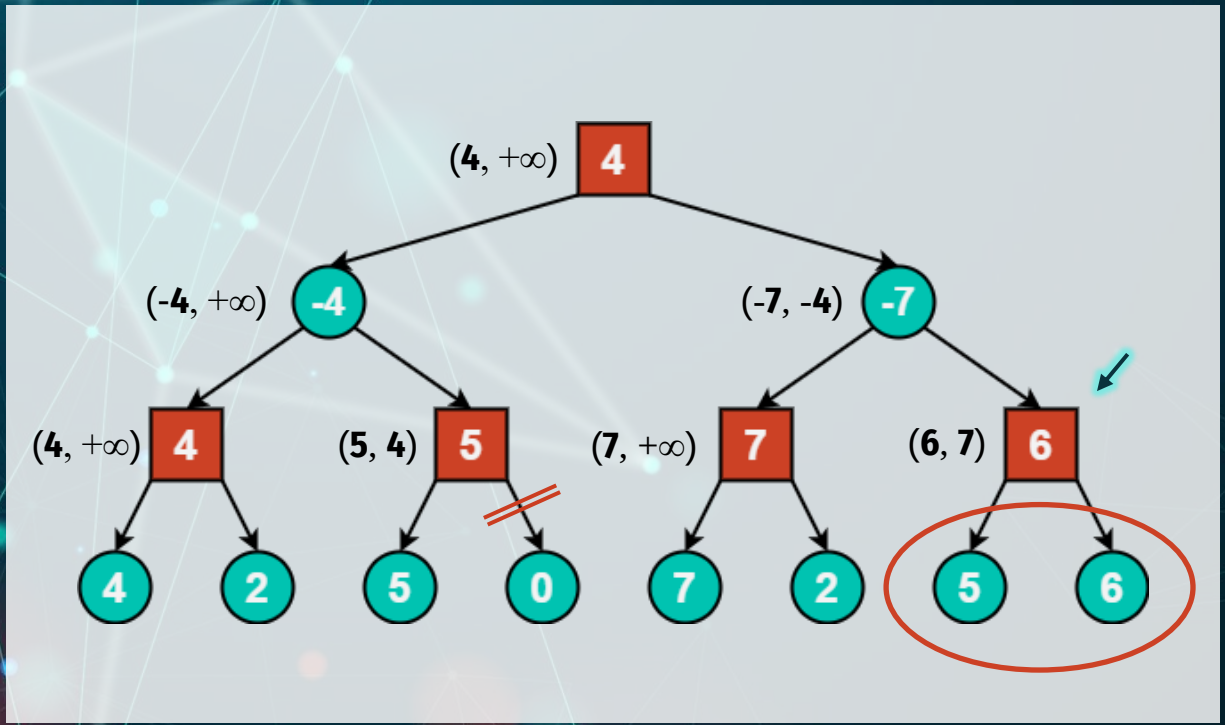
Zadatak 6 – Rešenje



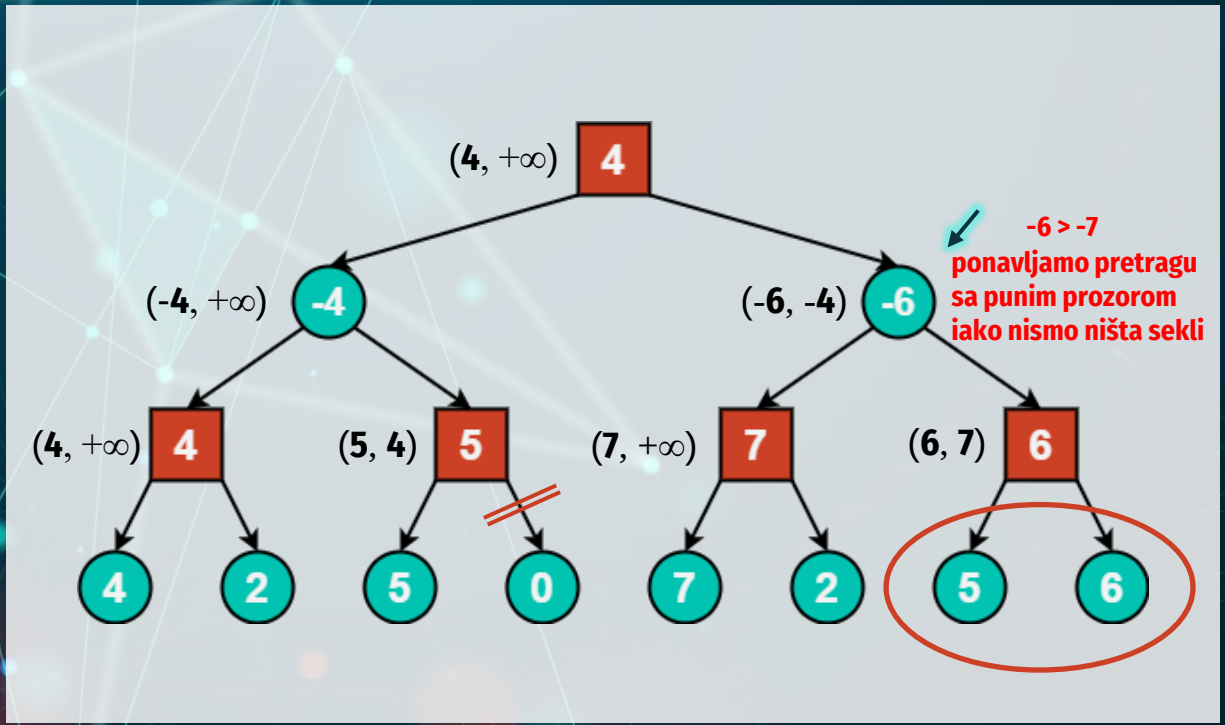
Zadatak 6 – Rešenje



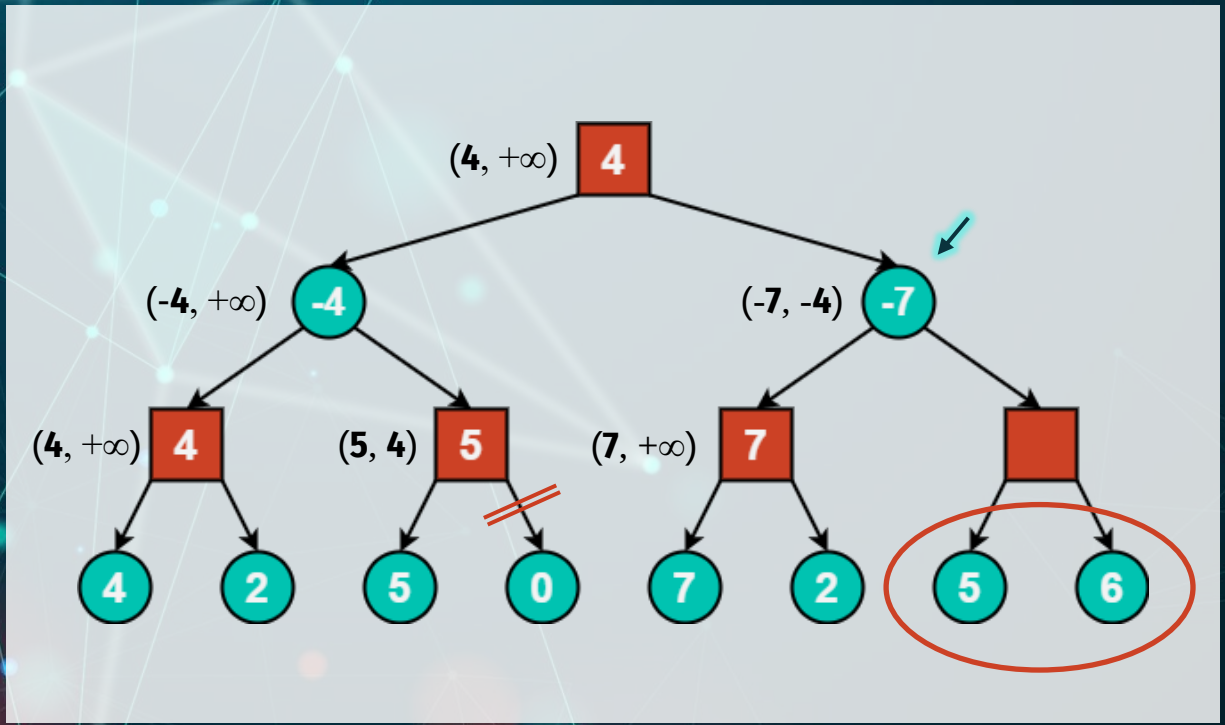
Zadatak 6 – Rešenje



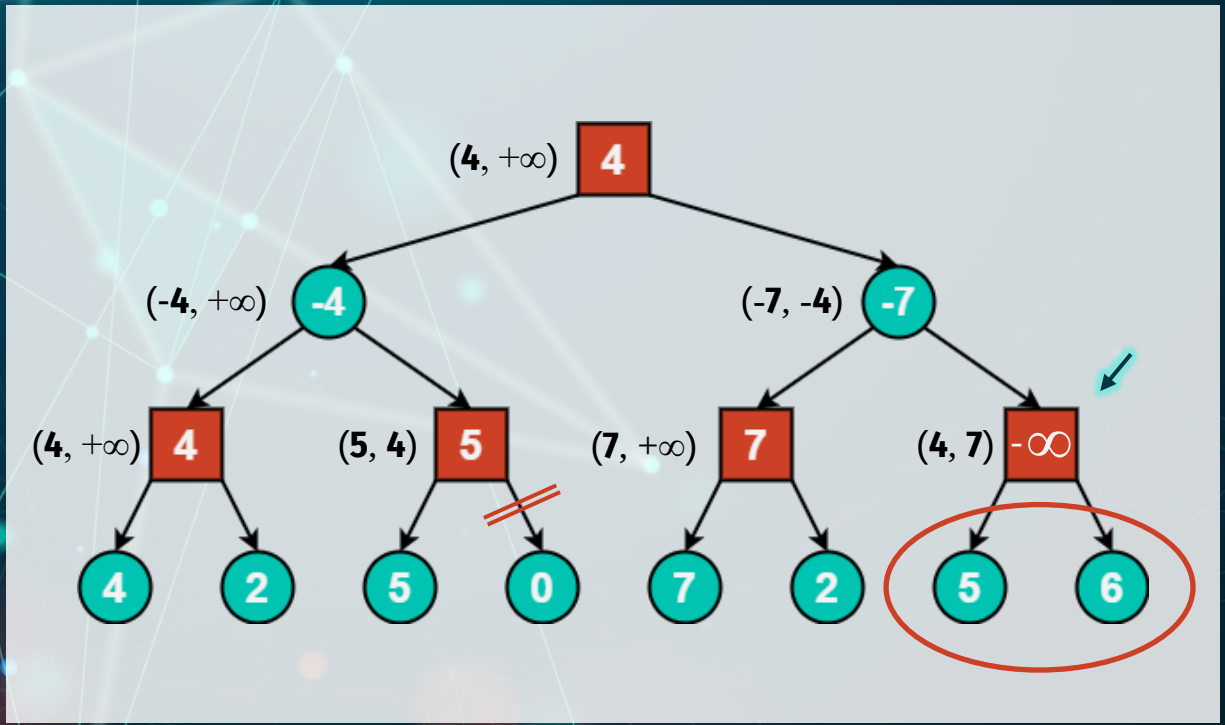
Zadatak 6 – Rešenje



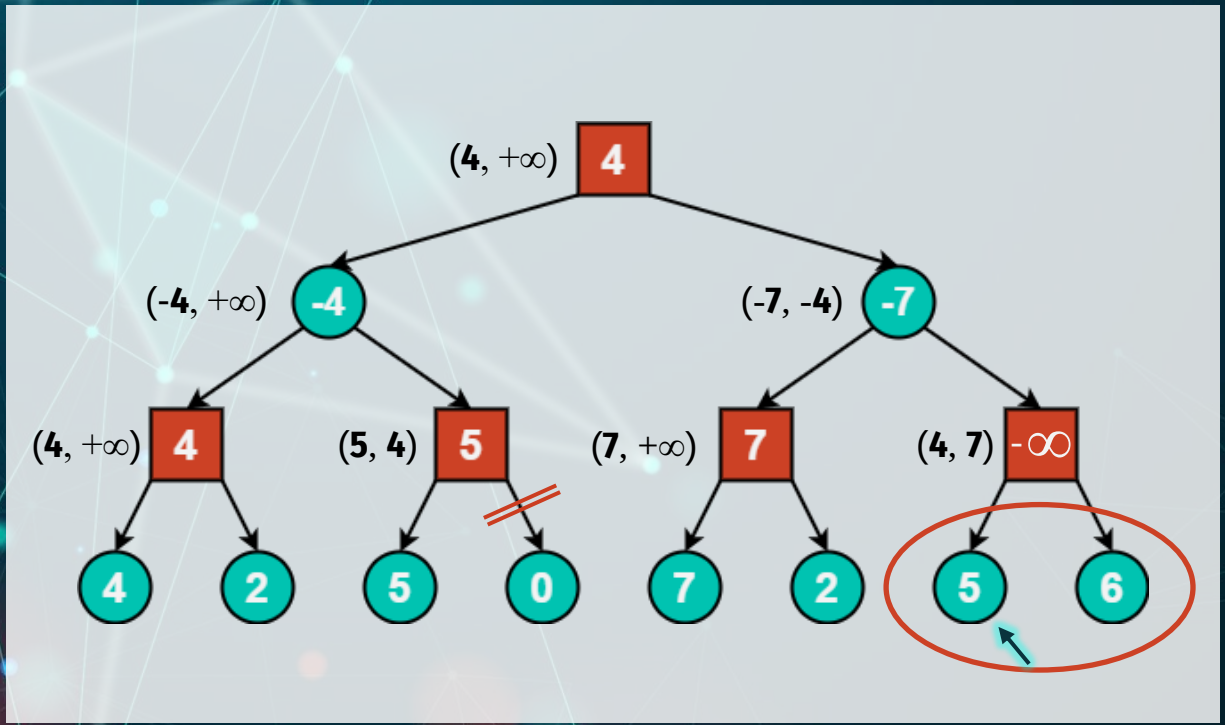
Zadatak 6 – Rešenje



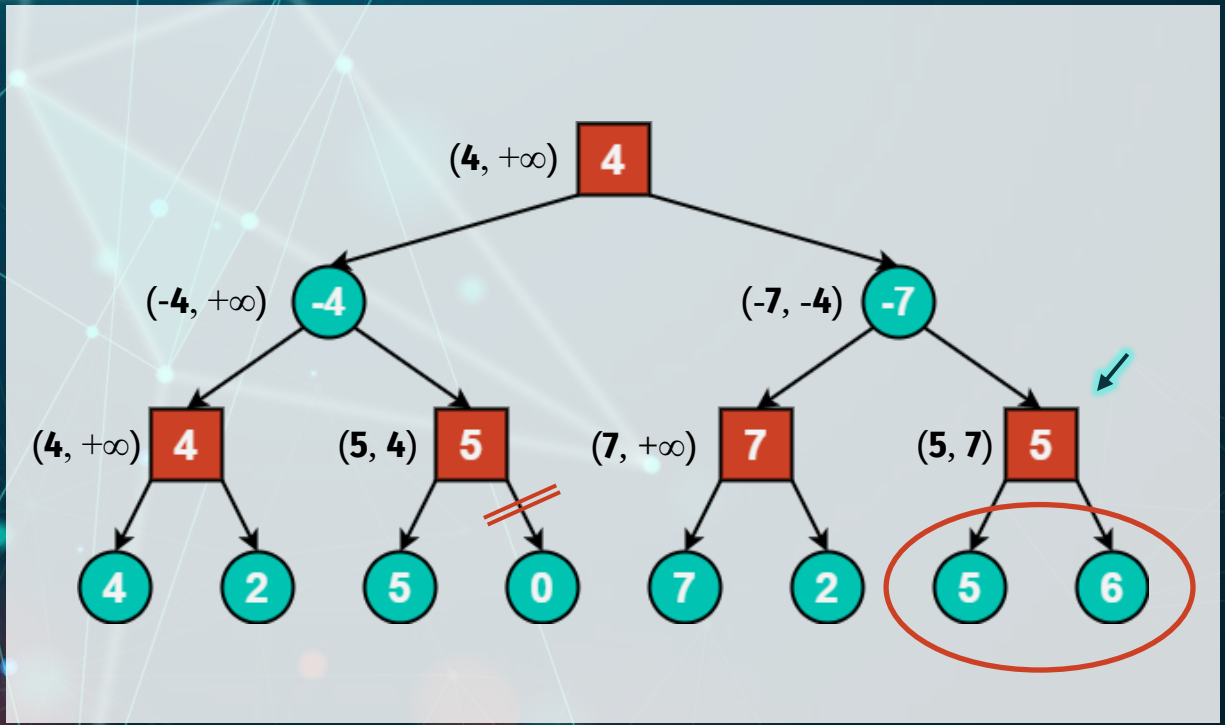
Zadatak 6 – Rešenje



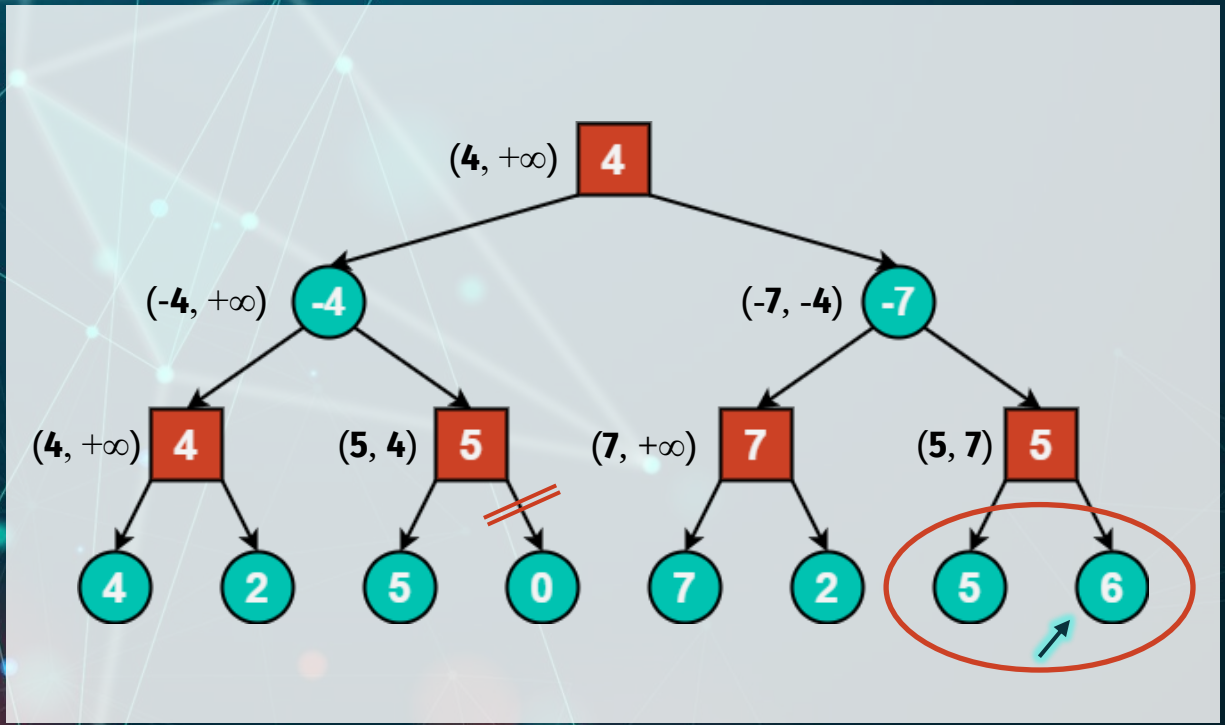
Zadatak 6 – Rešenje



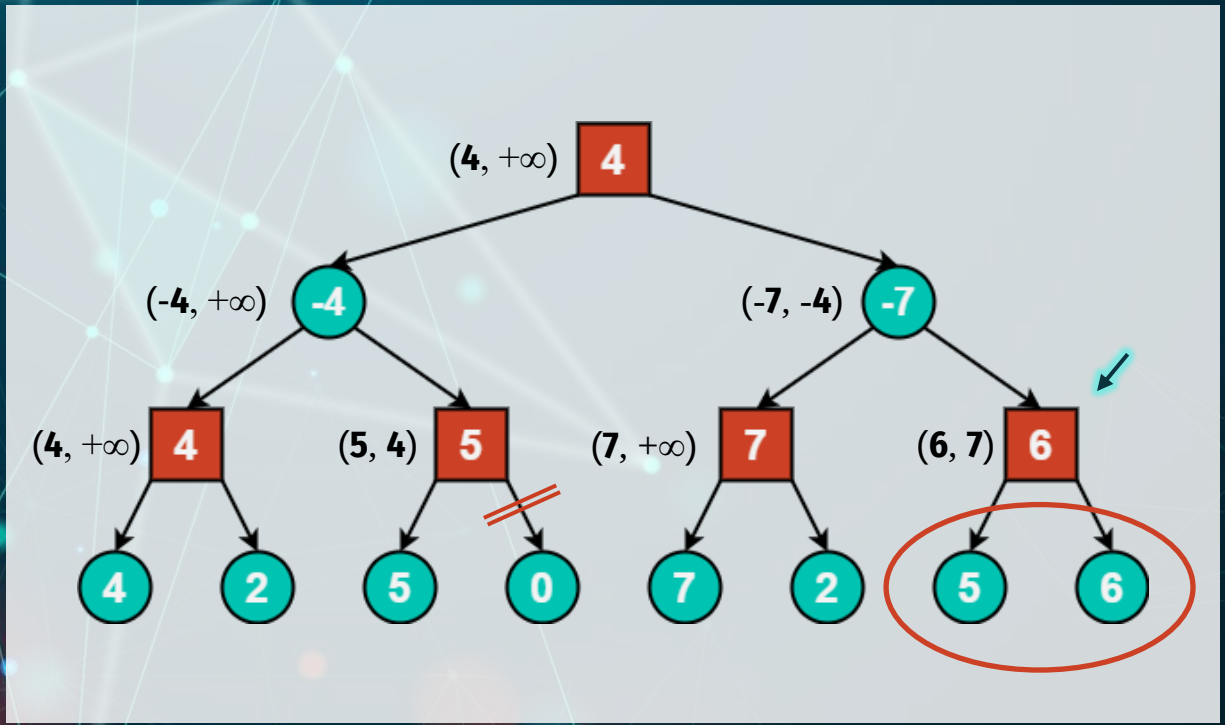
Zadatak 6 – Rešenje



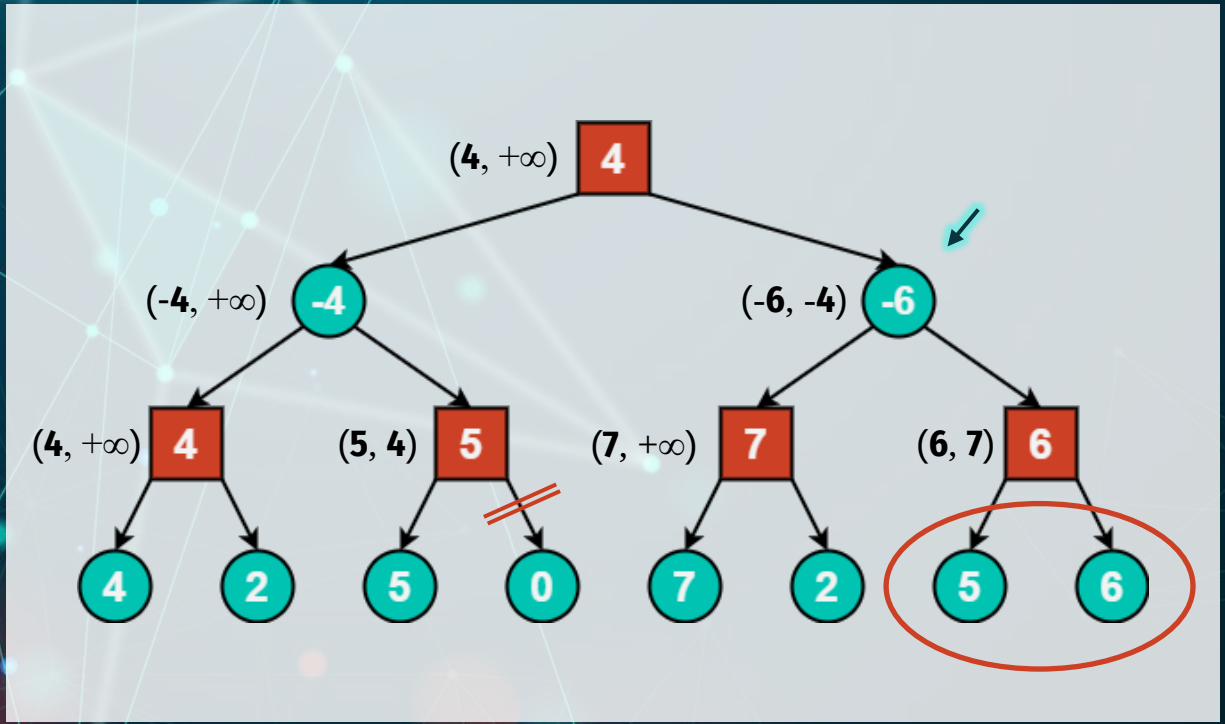
Zadatak 6 – Rešenje



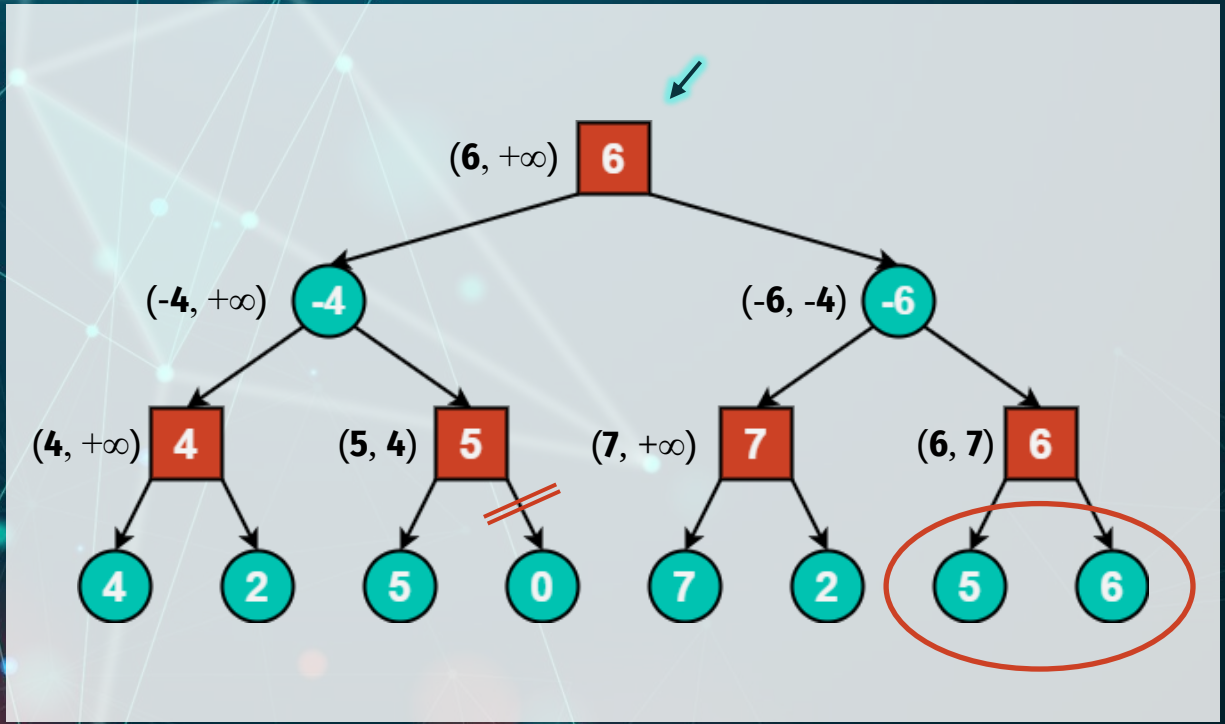
Zadatak 6 – Rešenje



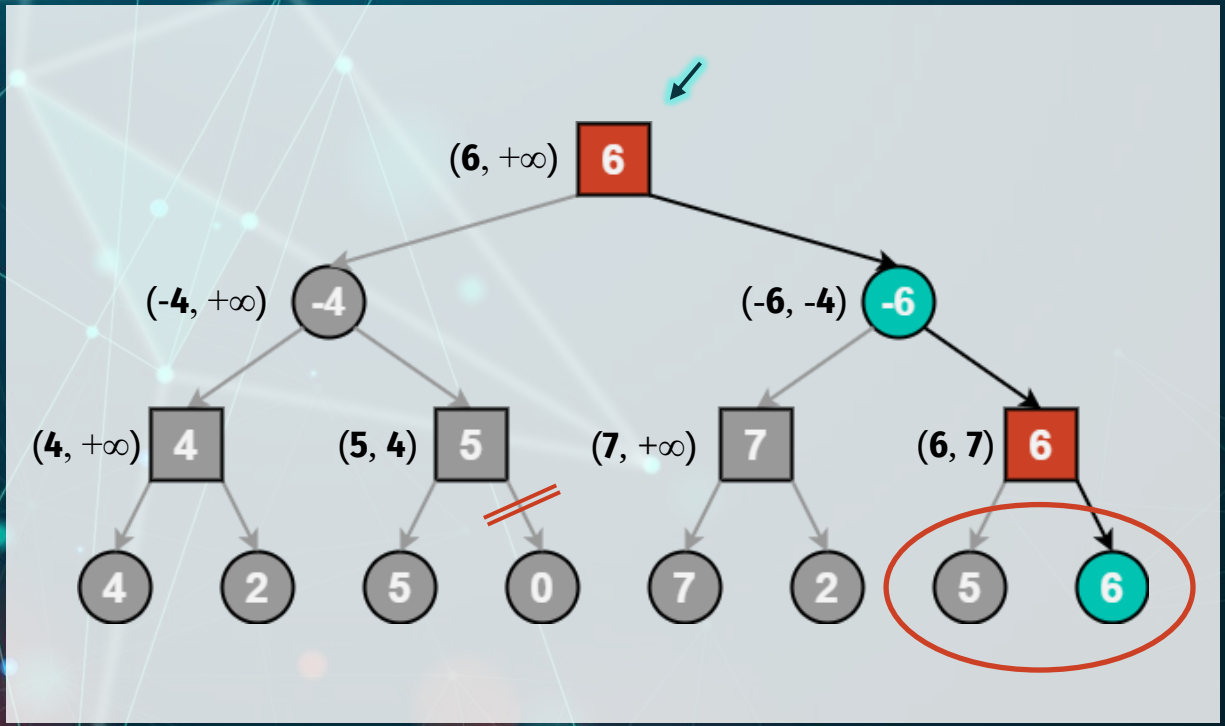
Zadatak 6 – Rešenje



Zadatak 6 – Rešenje

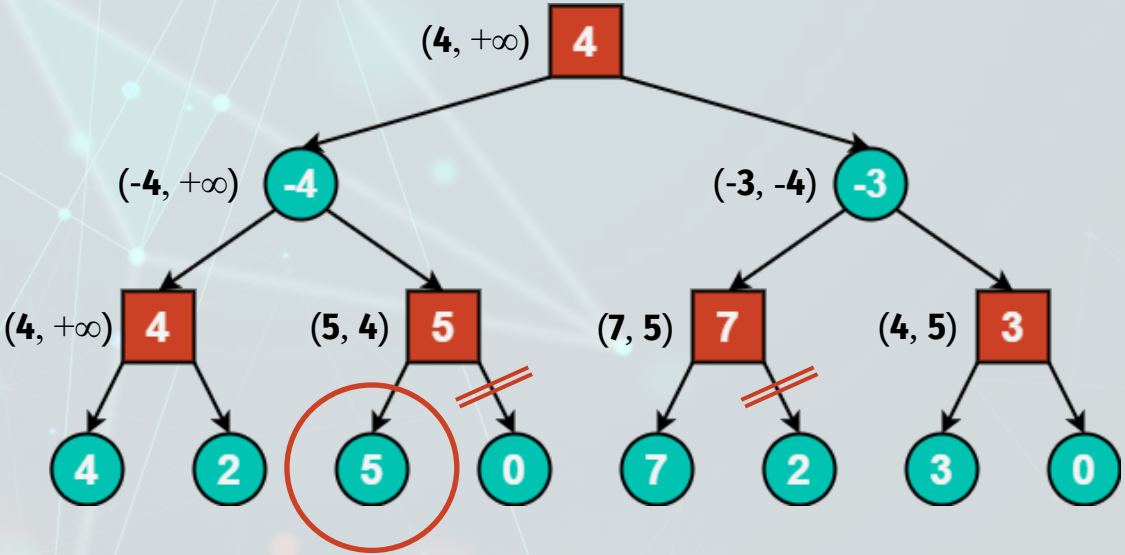


Zadatak 6 – Rešenje



Zadatak 6 – Rešenje

Za samostalnu vežbu! Šta bi se desilo ukoliko bismo u originalnom primeru izmenili vrednost označenog dela stabla u 3?

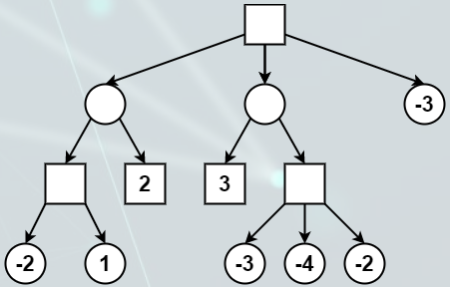


Zadatak za samostalnu vežbu – Borba veverica

★★★★☆

Sendi implementira pametnog protivnika za svoju novu igricu „Borba veverica“. Da bi bolje izučila algoritme koje treba da implementira, Sendi je odlučila da deo igre reši ručno. Na slikama se nalaze nepopunjena stabla čije je vrednosti potrebno popuniti u skladu sa algoritmom.

Dopuniti sve nedostajuće vrednosti u čvorovima ukoliko se za sledeću igru koristi Negamax algoritam. Na potezu (u korenu stabla) je igrač MAX.



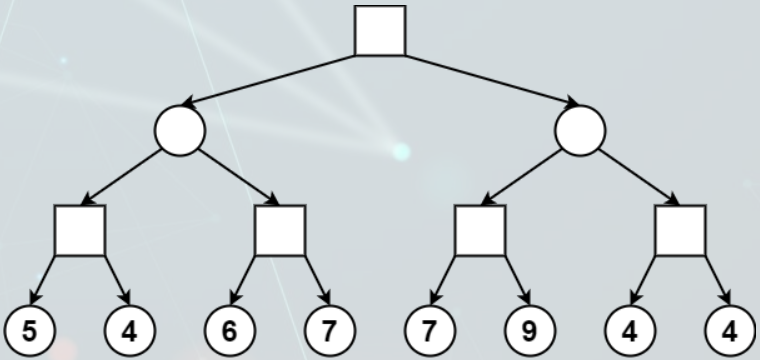
Kako treba izmeniti redosled grana u stablu sa slike tako da se dobije što veći broj odsecanja pri korišćenju alfa-beta poboljšanja Negamax algoritma. Nacrtati takvo stablo sa desne strane i označiti grane koje bi se u takvom stablu odsekle.

Zadatak za samostalnu vežbu – Borba veverica

★★★★☆

Dopuniti sve nedostajuće vrednosti u čvorovima ukoliko se za sledeću igru koristi Negascout algoritam. Pored svakog čvora, označiti alfa i beta granice odsecanja. Jasno označiti svaku granu koja je odsečena u toku algoritma i to: jednom crtom ukoliko je grana odsečena običnim alfa-beta odsecanjem, sa dve crte ukoliko je grana odsečena Negascout poboljšanjem. Na potezu (u korenu stabla) je igrač MAX.

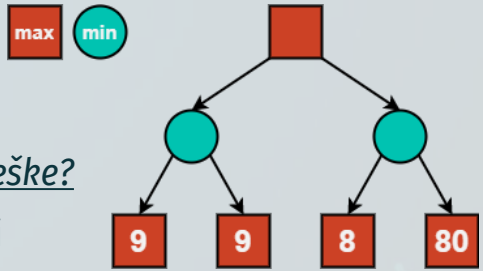
Da li je (i ukoliko jeste, na kom mestu) bilo potrebno vršiti ponovnu pretragu sa punim prozorom?



STOHASTIČKO OKRUŽENJE

Šta ako ne znamo šta će biti rezultat neke akcije?

- Bacanje kockica
- Bacanje novčića
- Podela karata

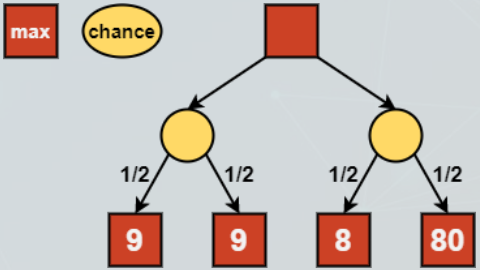


Šta ako protivnik nije optimalan ili pravi greške?

Ne želimo da maksimizujemo najgori mogući slučaj, već prosečni (optimistički pogled)

Takve situacije modelovaćemo uvođenjem čvorova šanse u stablo igre (*chance node*) kao novi nivo stabla ili umesto *min* čvorova.

Očekivane vrednosti čvorova šanse dobijaju se množenjem funkcije procene njihovih naslednika sa verovatnoćama takvih ishoda.



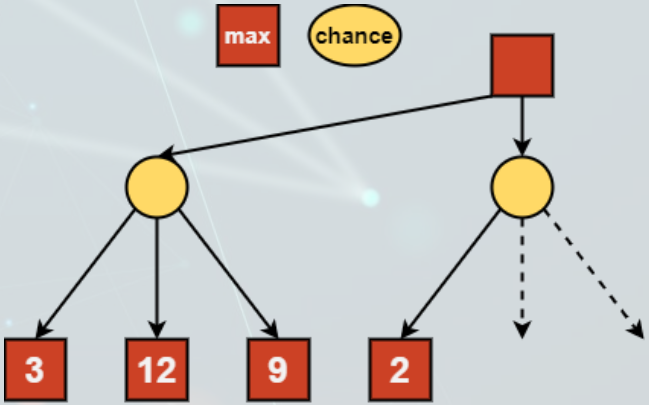
EXPECTIMAX ALGORITAM

```
def expectimax(node, player):
    if is_terminal_node(node):
        return node_evaluation(node)
    score = -math.inf if player == Player.MAX else (+math.inf if y == Player.MIN else 0)

    if player == Player.MAX:
        for succ in node.successors():
            score = max(score, expectimax(node, switch(player)))
        return score
    elif player == Player.MIN:
        for succ in node.successors():
            score = min(score, expectimax(node, switch(player)))
        return score
    else:
        for succ in node.successors():
            prob = probability(succ)
            score += prob * expectimax(node, switch(player))
        return score
```

EXPECTIMAX ALPHA-BETA PRUNING

U opštem slučaju nije moguće primeniti algoritam alfa-beta odsecanja na expectimax pretragu, jer vrednost *chance* čvorova ne predstavlja maksimizaciju najgoreg mogućeg ishoda već težinsku sumu svojih naslednika (ako svi naslednici nisu podjednako verovatni).



Zadatak 7 – Expectimax

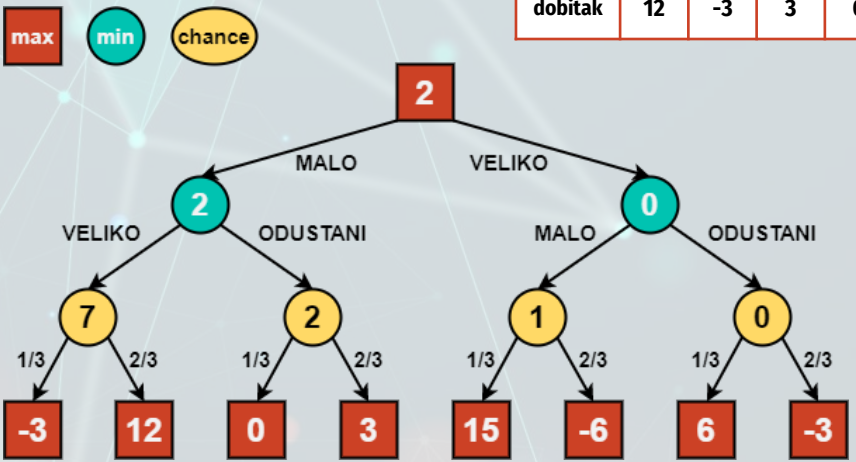
★★★☆☆

Data je sledeća igra sa kockicom za dva igrača. Prvi igrač predviđa ishod bačene kockice. Na raspolaganju su mu sledeće odluke: **pašće mali broj (1-4)** i **pašće veliki broj (5-6)**. Nakon prvog igrača, drugi igrač takođe učestvuje u predviđanju ishoda bačene kockice tako što bira odluku koju nije odabrao prvi igrač ili odustaje. Na slici je dat očekivani dobitak za prvog igrača za sve moguće odluke oba igrača. Nacrtati kompletno stablo igre, a zatim primenom **expectimax** algoritma odrediti funkciju procene za svaki čvor ukoliko su oba igrača racionalna.

Igrač 1	M	M	M	M	V	V	V	V
Igrač 2	V	V	O	O	M	M	O	O
Kocka	M	V	M	V	M	V	M	V
dobitak	12	-3	3	0	-6	15	-3	6

Zadatak 7 – Rešenje

Igrač 1	M	M	M	M	V	V	V	V
Igrač 2	V	V	O	O	M	M	O	O
Kocka	M	V	M	V	M	V	M	V
dobitak	12	-3	3	0	-6	15	-3	6

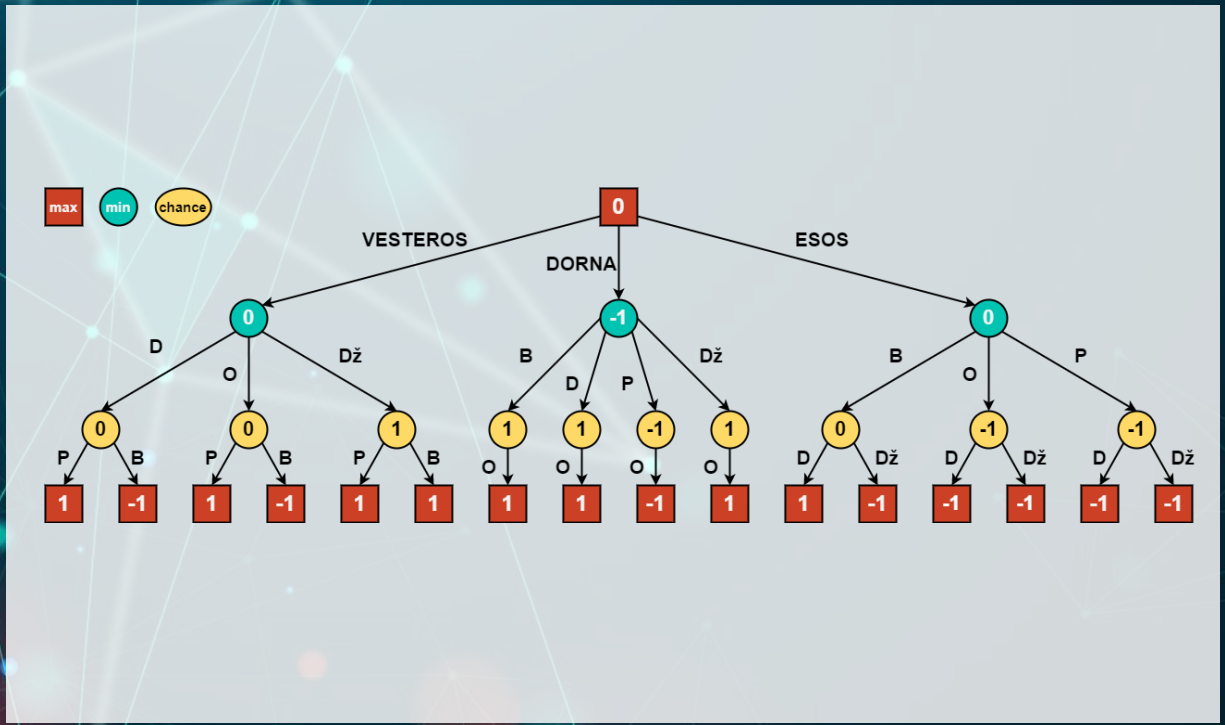


Zadatak za samostalnu vežbu – Duel borba

★★★★☆

Lord Tirion je osumnjičen za ubistvo svog sestrića (u isto vreme i kralja) Džofrija od strane njegovog oca, Tajvina. Sudi mu se kroz duel borbu pa Tajvin i Tirion biraju svoje borce. Ukoliko Tirionov borac bude poražen Tirion će biti proglašen krivim, a u suprotnom će biti oslobođen. Tajvin ima važnija posla od izbora borca pa samo bira oblast iz koje će njegov borac biti. Nakon toga Tirion može samo izabrati borca koji nije iz oblasti koju je izabrao Tajvin. Nakon Tirionovog izbora, Tajvin prepušta pisaru izbor borca. Pisar je laik za borbe, te na uniformno slučajan način bira borca iz izabrane Tajvinove oblasti. Pobeđuje borac koji je jači. Borci koji se mogu zvati su navedeni po strogo opadajućoj jačini (oblast iz koje dolazi je u zagradi): Planina (Vesteros), Oberin (Dorna), Dario (Esos), Bron (Vesteros), Džora (Esos). Nacrtati stablo igre, popuniti sve vrednosti u njemu i označiti na kom se nivou stabla nalazi koja vrsta čvorova. Uzeti da se čvorovi ekspanduju po sledećem poretku oblasti: Vesteros, Dorna, Esos, odnosno boraca: Bron, Dario, Oberin, Planina, Džora.

Zadatak za samostalnu vežbu – Rešenje



Zadatak za samostalnu vežbu – Rizik

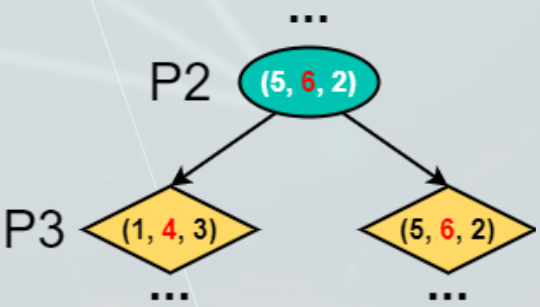
★★★★☆

Tom je predstavio Džeriju svoju novu igru u kojoj je cilj da se sakupi što više poena, ali da se ne pređe ograničenje od 3 poena. U svakom potezu, igrač ima na raspolaganju dve opcije: da nastavi dalje igru ili da odustane od dalje igre i time unovči do tada osvojene poene. Ukoliko nastavi igru, baca se trostrana kockica čiji su mogući ishodi 1, 2 ili 3 i ishod kockice se dodaje na igračeve trenutne poene, nakon čega igrač ponovo ima opciju da nastavi ili prekine igru, i tako u krug. Cilj igre je unovčiti što više poena koji ne prelaze 3. Igra se završava kada igrač odustane od igre i time unovči do tada ostvarene poene ili kada se sakupi 3 ili više poena. Ukoliko igrač sakupi više od 3 poena, tada se njegovi poeni menjaju negativnom razlikom između sakupljenog broja poena i broja 3 (npr. 5 poena = -2 poena). Džeri je odlučio da predstavi igru Expectimax algoritmom kako bi procenio svoj najbolji potez. Nacrtati stablo igre i popuniti sve vrednosti čvorova.

IGRE U VIŠE IGRAČA – MAX^n ALGORITAM

U igrama sa više igrača potrebno je pamtiti dobitke za svakog igrača unutar torke koja je pridružena svakom stablu igre, gde je *i*-ta vrednost dobitak za *i*-tog igrača.

Igrači u svojim potezima biraju onu torku koja na *i*-toj poziciji ima veću vrednost. To znači da će se u internim čvorovima stabla igre uvek vršiti maksimizacija nad nekim elementom torke, i najbolja torka propagirati na sledeći nivo.



MAX^n ALGORITAM

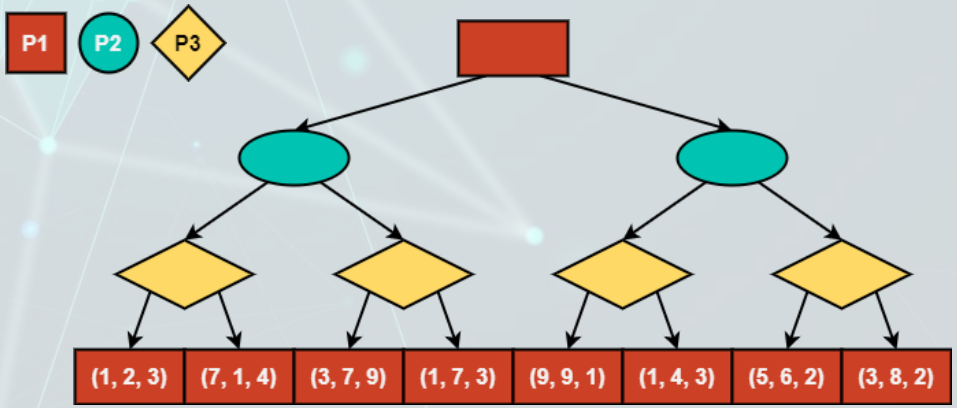
```
def maxn(node, player):
    if is_terminal_node(node):
        return node_evaluation_list(node)

    score_list = [-math.inf for i in range(get_player_count())]
    i = player.get_index()
    for succ in node.successors():
        child_score_list = maxn(node, get_next(player))
        score_list = score_list if score_list[i] >= child_score_list[i] else child_score_list)
    return score_list
```

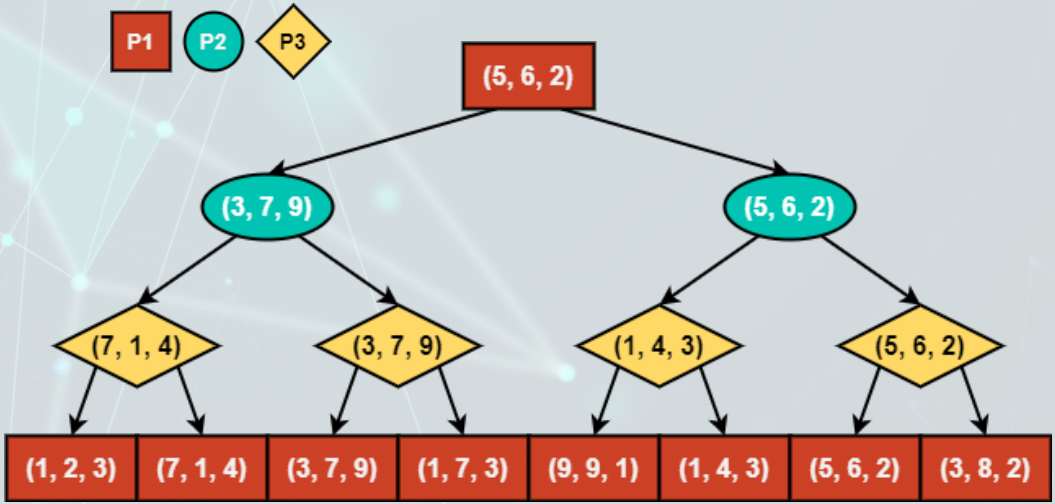
Zadatak 8 – Max^N

★★☆☆☆

Dato je kompletno stablo igre sa naznačenim dobicima za svakog igrača. Primenom max^N algoritma odrediti funkciju procene za svaki čvor, konačan ishod igre i sekvencu poteza koja će biti odigrana do kraja ukoliko su svi igrači racionalni.



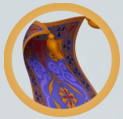
Zadatak 8 – Rešenje



Zadatak za samostalnu vežbu – Zlatnici

★★★★★

Duh, Abu i Ćilim igraju sekvencijalnu igru „Zlatnici“. Svaki igrač počinje igru sa dva zlatnika u ruci, a zajednički talon sa zlatnicima je prazan. Igrač na potezu ima opciju da doda zlatnik iz ruke na zajednički talon ili da ukrade sve zlatnike sa talona. Zlatnici sa talona mogu da se ukradu samo ukoliko se na talonu nalazi barem jedan zlatnik. Ukradeni zlatnici se isključuju iz igre i čuvaju pored igrača koji ih je ukrao. Svaki igrač sme da se odluči za krađu zlatnika maksimalno jednom u toku partije. Igra se završava kada igrač koji je na potezu nema šta da odigra (bez obzira da li ostali igrači imaju dostupne poteze koje mogu da odigraju). Nakon završetka igre svaki zlatnik koji je eventualno ostao na talonu se vraća u ruku igraču koji ga je tu stavio. Igračev dobitak se računa kao zbir ukradenih zlatnika i zlatnika koji su igraču ostali u ruci. Igru započinje Duh, nakon kojeg igra Abu, a zatim Ćilim. Nacrtati stablo igre i popuniti vrednosti svih čvorova koristeći Max^N algoritam. U slučaju da igraču više različitih poteza donosi jednaku dobit, igrač daje prednost potezu UKRADI.



Zadatak za samostalnu vežbu – Zlatnici

★★★★★

Koji potezi će biti odigrani i šta će biti dobitak svakog od igrača nakon završetka partije ukoliko svi igrači igraju racionalno?

Šta bi bio dobitak svakog igrača nakon završetka partije ukoliko su prva tri poteza u igri DODAJ, DODAJ, DODAJ, a na dalje svi igrači igraju racionalno?

Navesti redosled odigranih poteza i dobitak svakog od igrača nakon završetka partije u kojoj su Duh i Ćilim su sklopili tajno savezništvo i dogovorili se da na kraju partije podele dobitak.

PITANJA?

http://ri4es.etf.rs/