
# Structuri de date și algoritmi

## MiniMax

Lect. Dr. Ing. Cristian Chilipirea – cristian.chilipirea@mta.ro

# Reminder backtracking

# Problemă: Varza, Capra și Lupul

**Lup**
mănâncă
**capra**

**Capra**
mănâncă
**varza**

Scop: mutarea unul câte unul pe mal opus fără să se mănânce

# Problemă: Varza, Capra și Lupul

**NU**

**Lupul** mănâncă **capra**

# Problemă: Varza, Capra și Lupul

**NU**

**Capra**
mănâncă
**varza**

# Problemă: Varza, Capra și Lupul

# Problemă: Varza, Capra și Lupul

# Problemă: Varza, Capra și Lupul

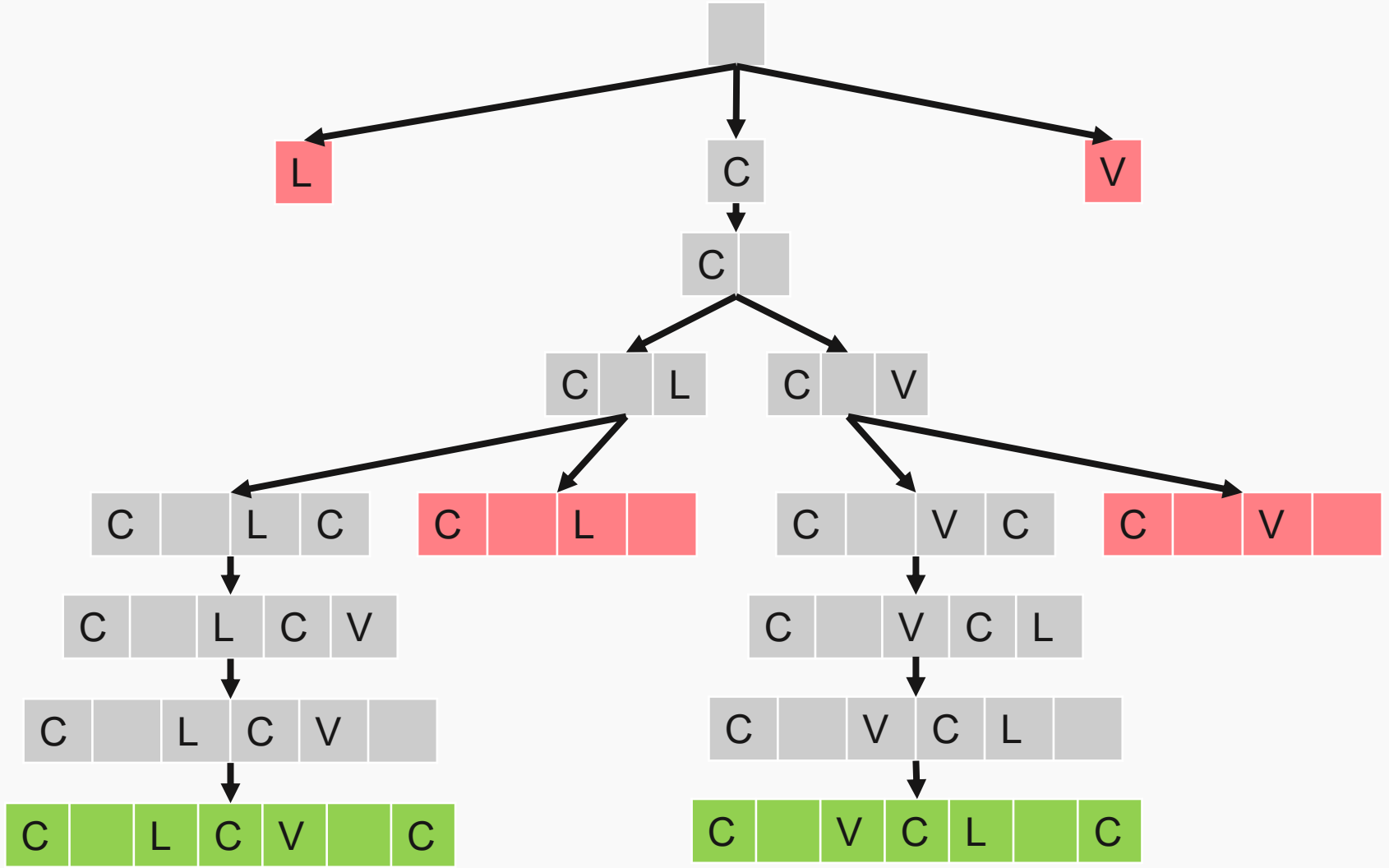
# Problemă: Varza, Capra și Lupul

# Problemă: Varza, Capra și Lupul

# Problemă: Varza, Capra și Lupul

# Problemă: Varza, Capra și Lupul

# Problemă: Varza, Capra și Lupul



# Abordarea Dijsktra – varză capră și lup

https://www.youtube.com/watch?v=0kXjl2e6qD0

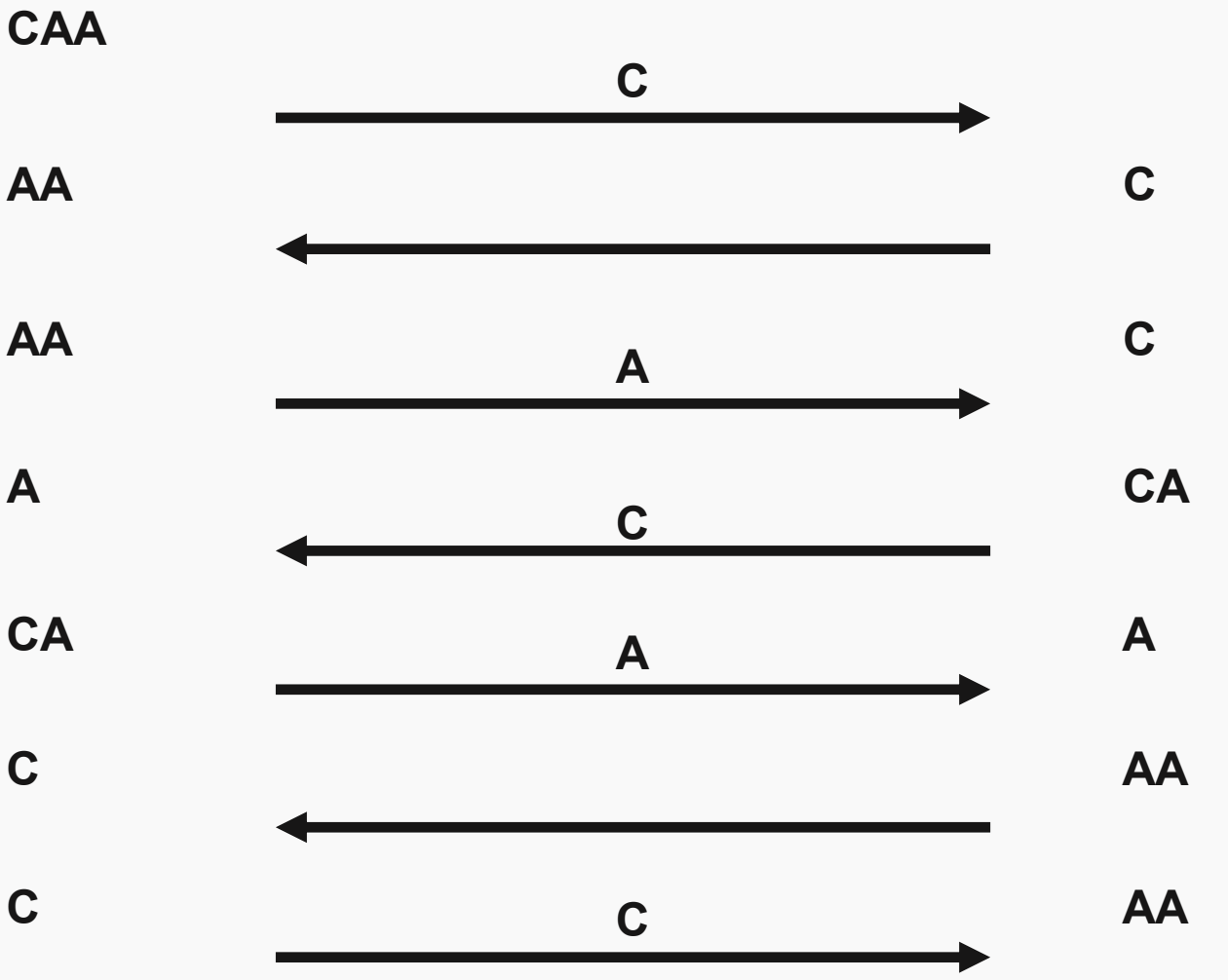
Scoate toate detaliile nesemnificative din problemă

Contează dacă lupul mănâncă capra sau dacă invers capra mănâncă lupul?

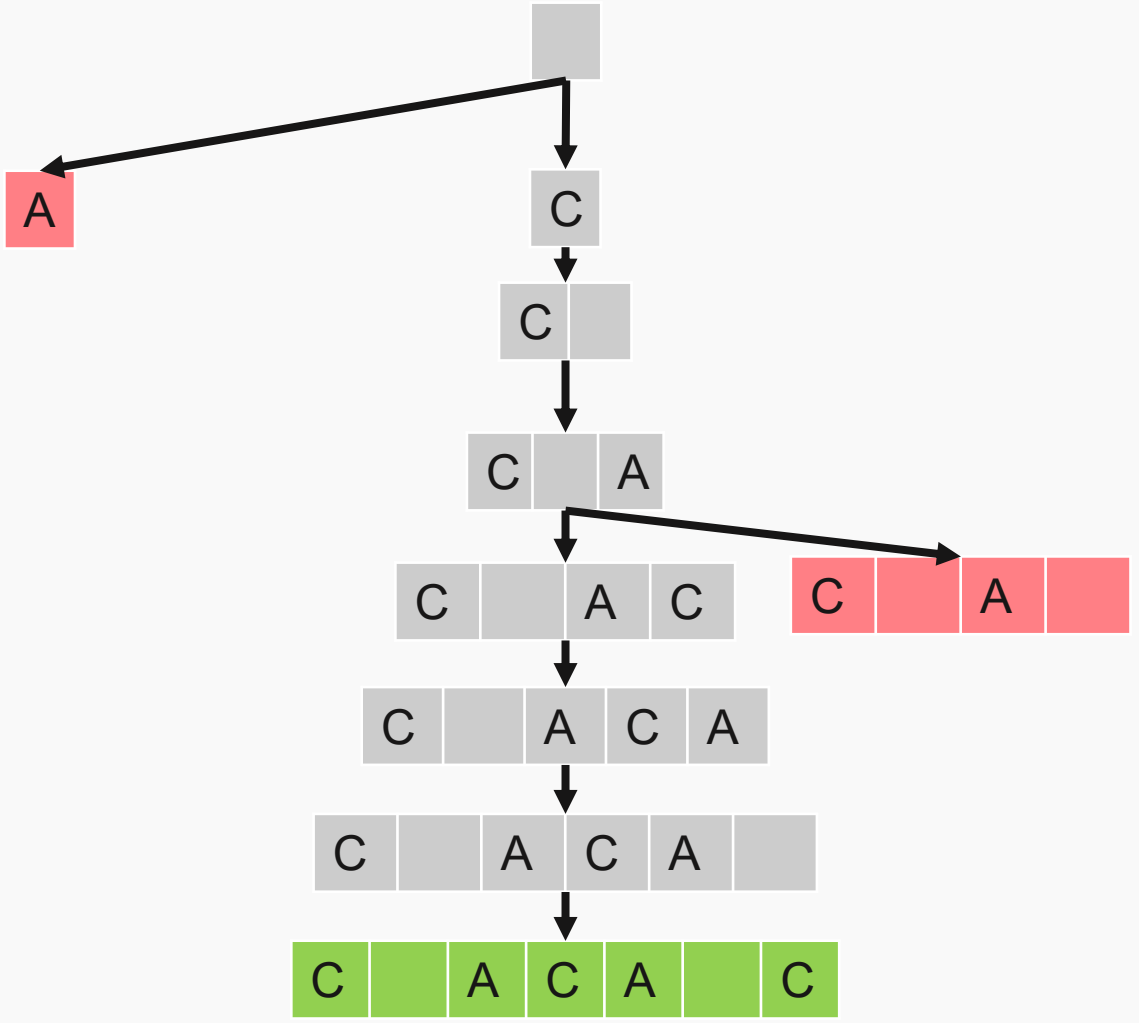
Contează dacă capra mănâncă varză sau dacă invers lupul mănâncă capra?

Dacă NU atunci nu e diferență între lup și varză.

# Capră și restul



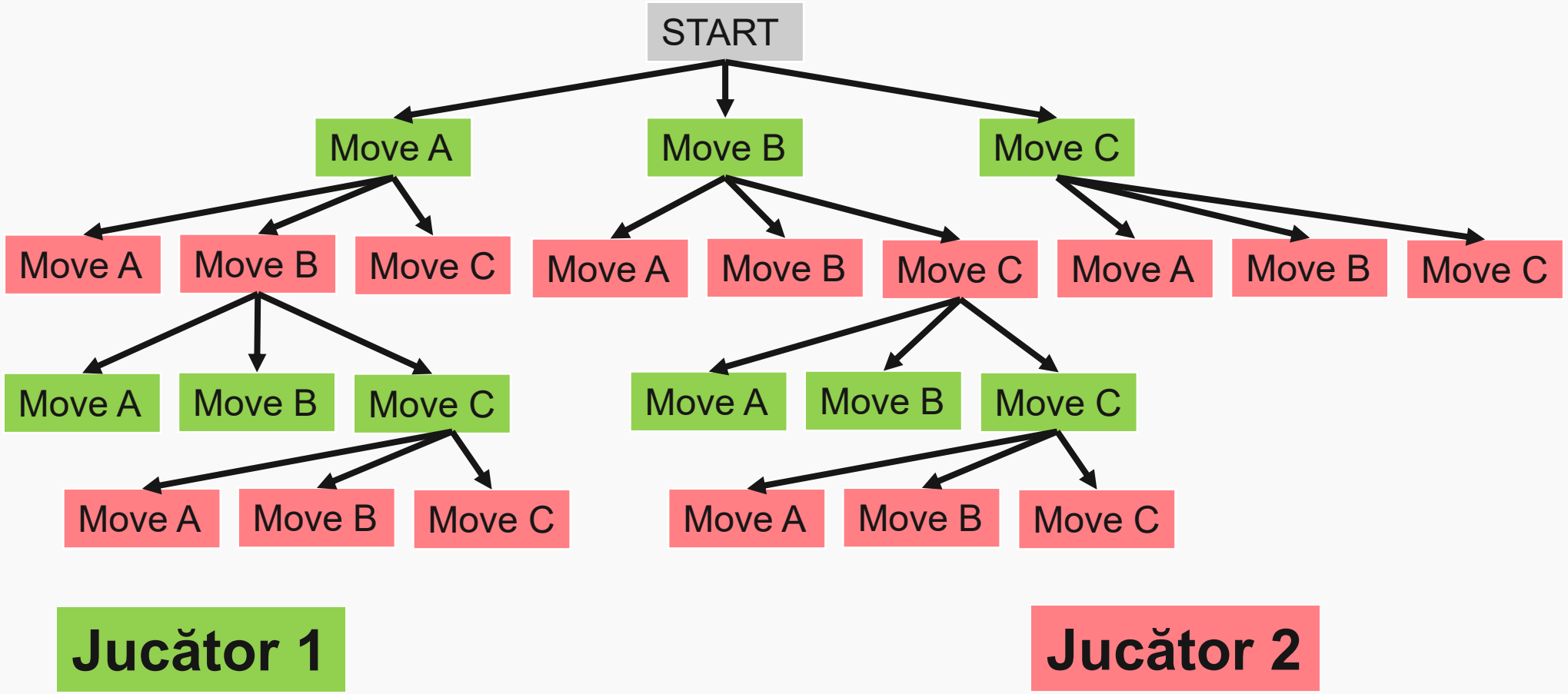
# Problemă: Varza, Capra și Lupul



# MiniMax

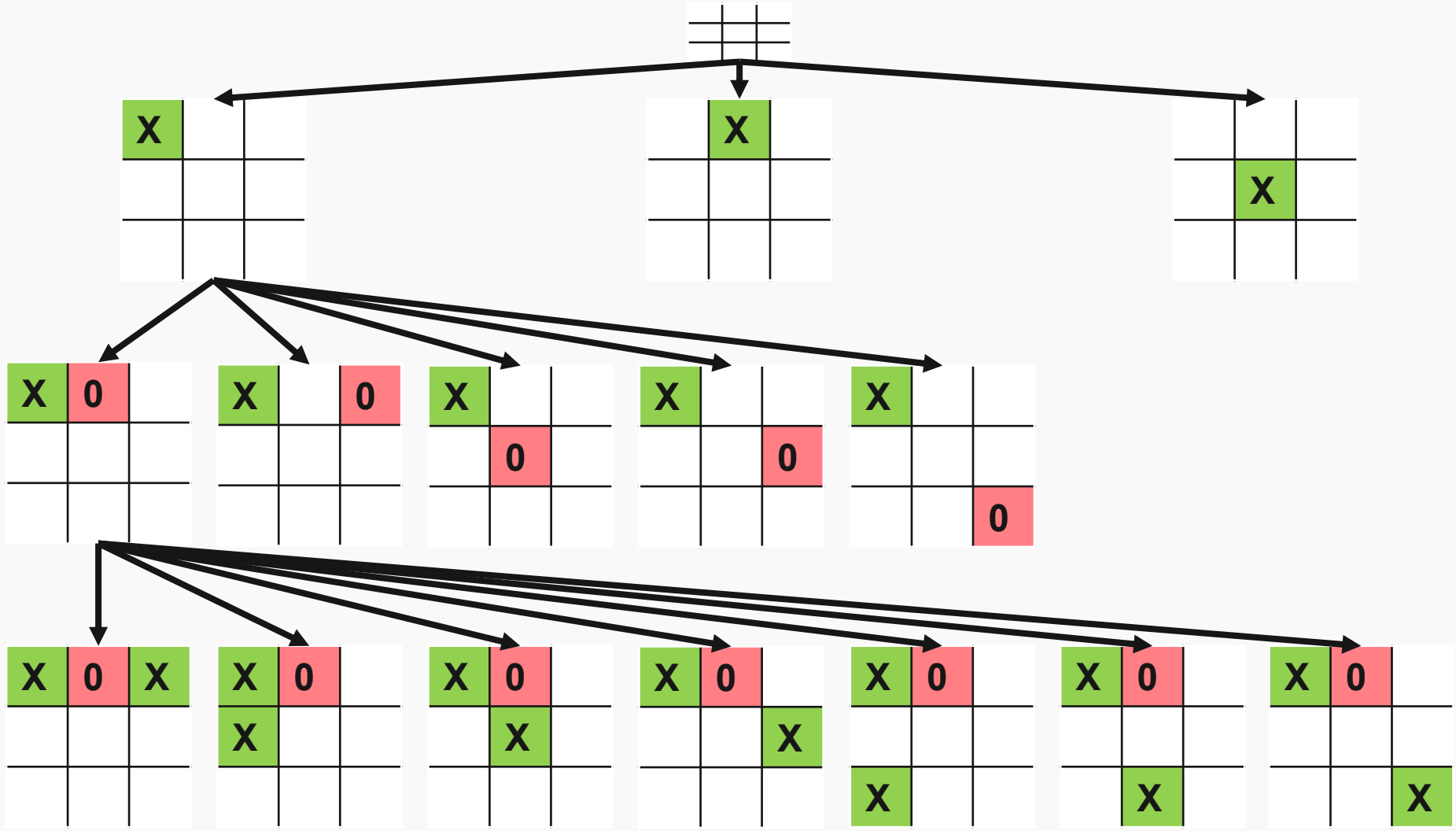
- Ideea de la backtracking cu construcția arborelui de stări.
- Jocuri competitive. – Vrem algoritm “Inteligență artigicială” care să câștige.
- Ce face adversarul?

# MiniMax - Idee




Pe nivelele pare sunt mutări ale unui jucător. Pe nivele impare ale altui jucător.

# MiniMax – X și 0



# MiniMax

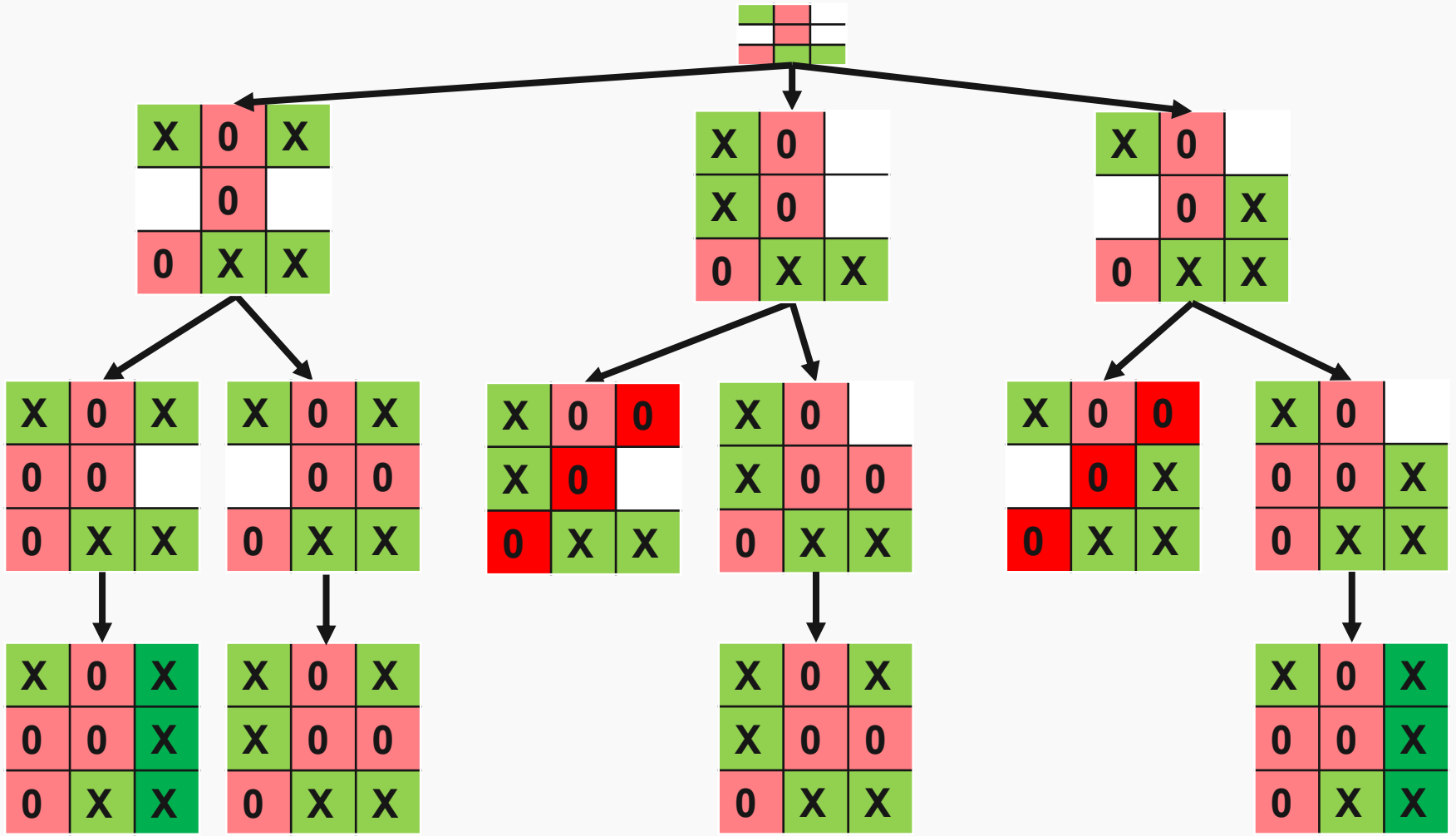
- Nu mai este suficient sa aplic backtracking.

- Chiar dacă găsesc un set de mutări prin care să câștig adversarul va alege mutări care să facă opusul.

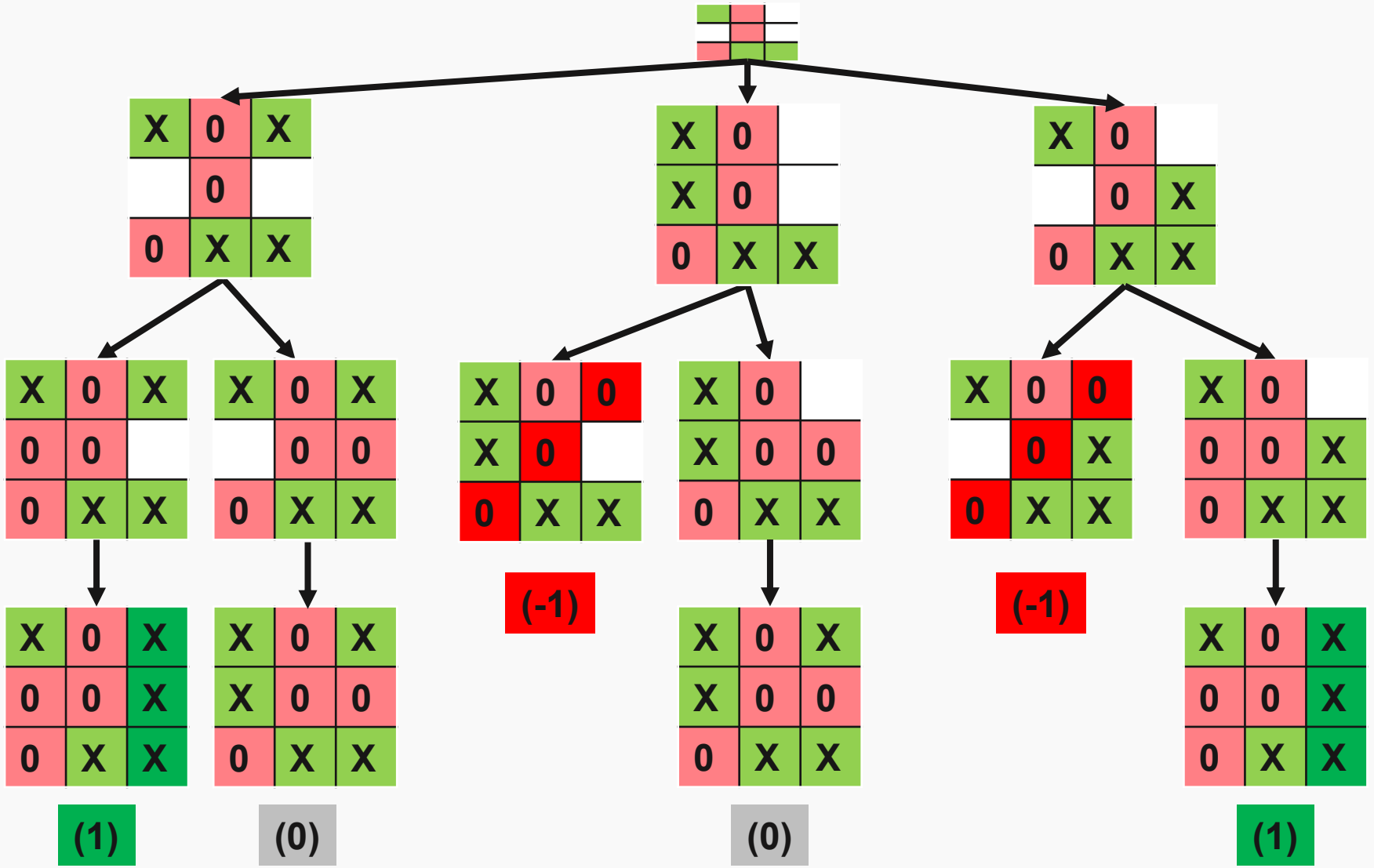
**La nivelele unde eu aleg, fac mutări care să maximizeze șansele de câștig.**

**La nivelele unde alege adversarul, acesta probabil va încerca să îmi minimizeze șansele de câștig. Deci așa presupun că va face.**

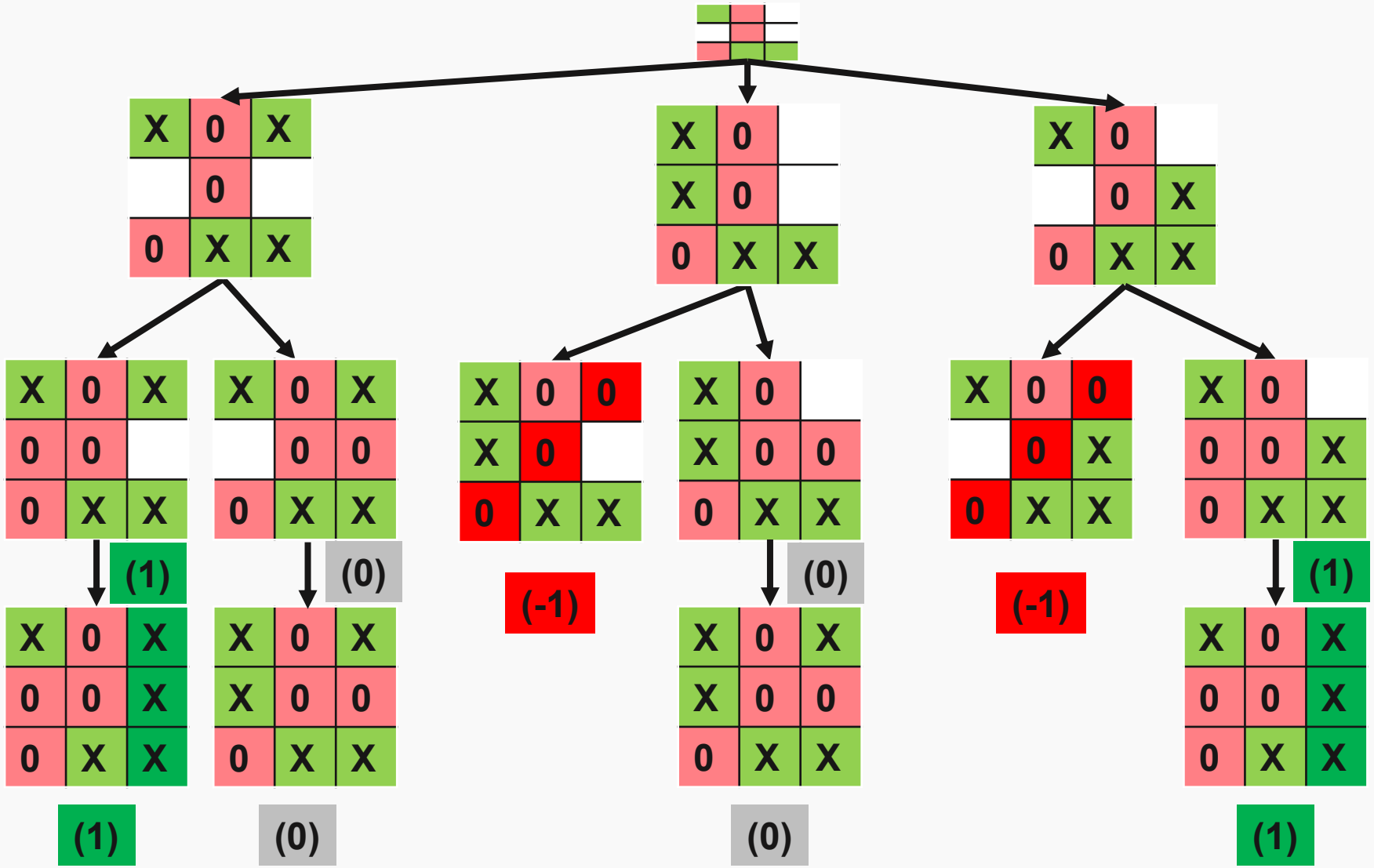
# MiniMax – X și 0



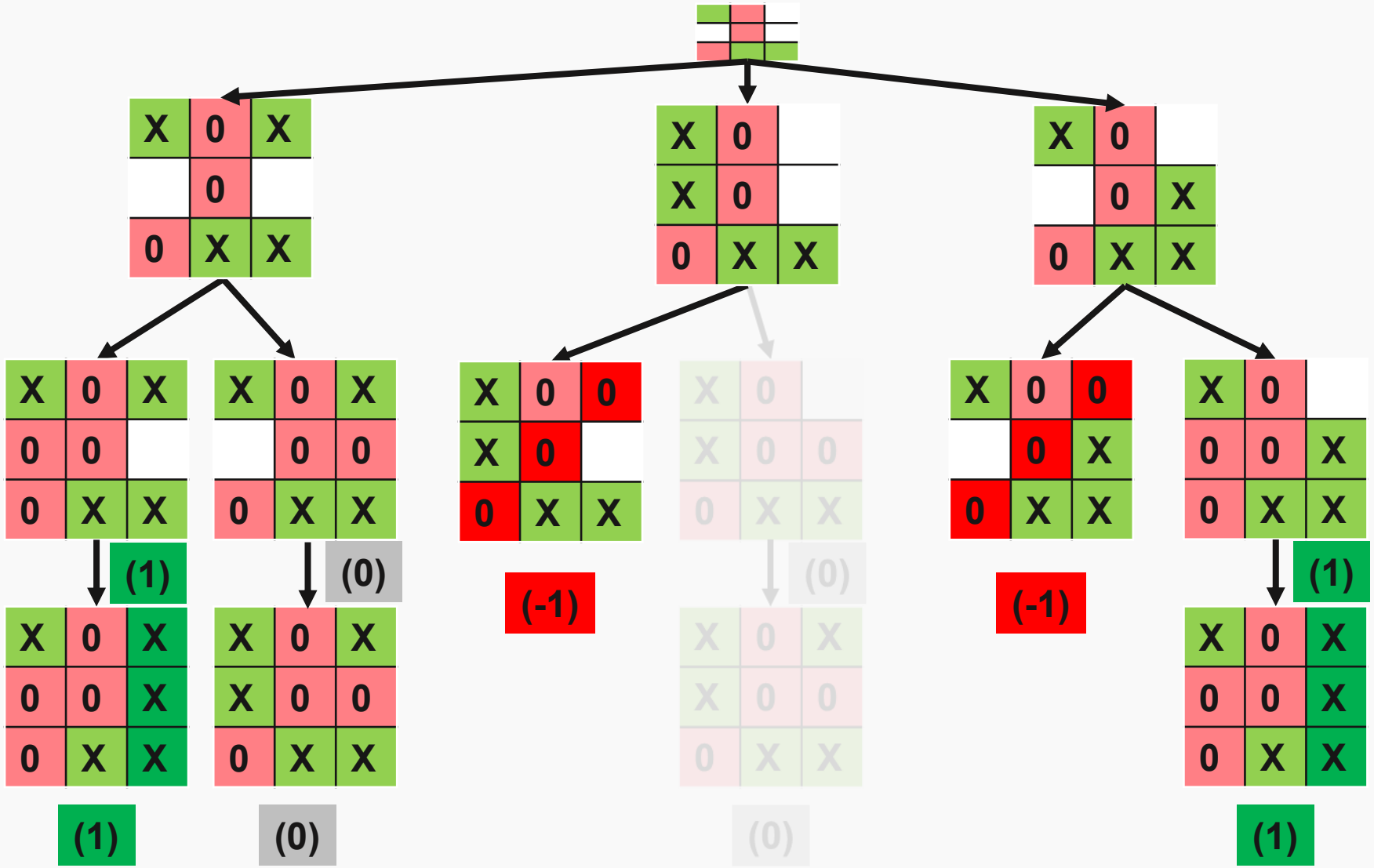
# MiniMax – X și 0



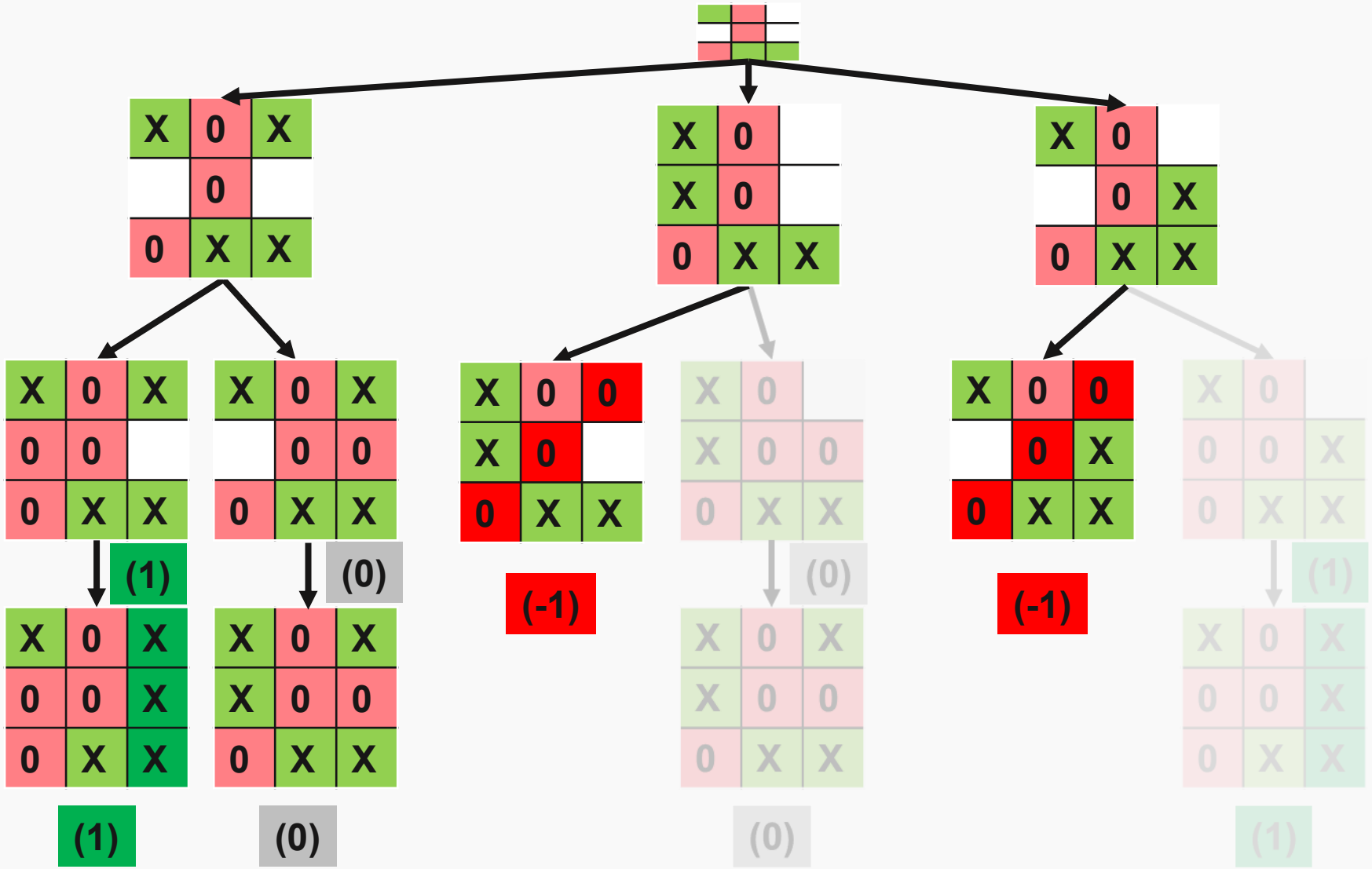
# MiniMax – X și 0



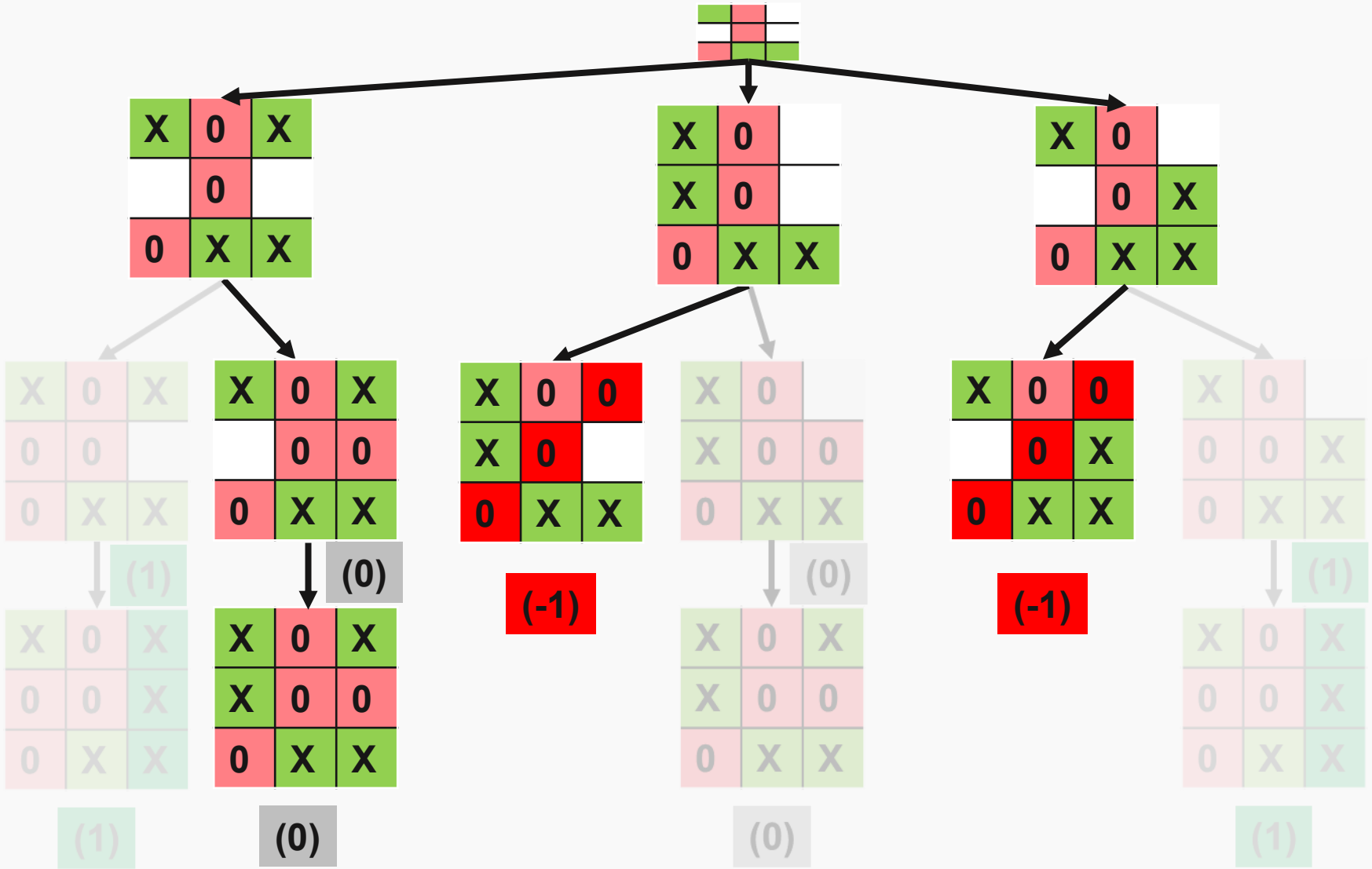
# MiniMax – X și 0



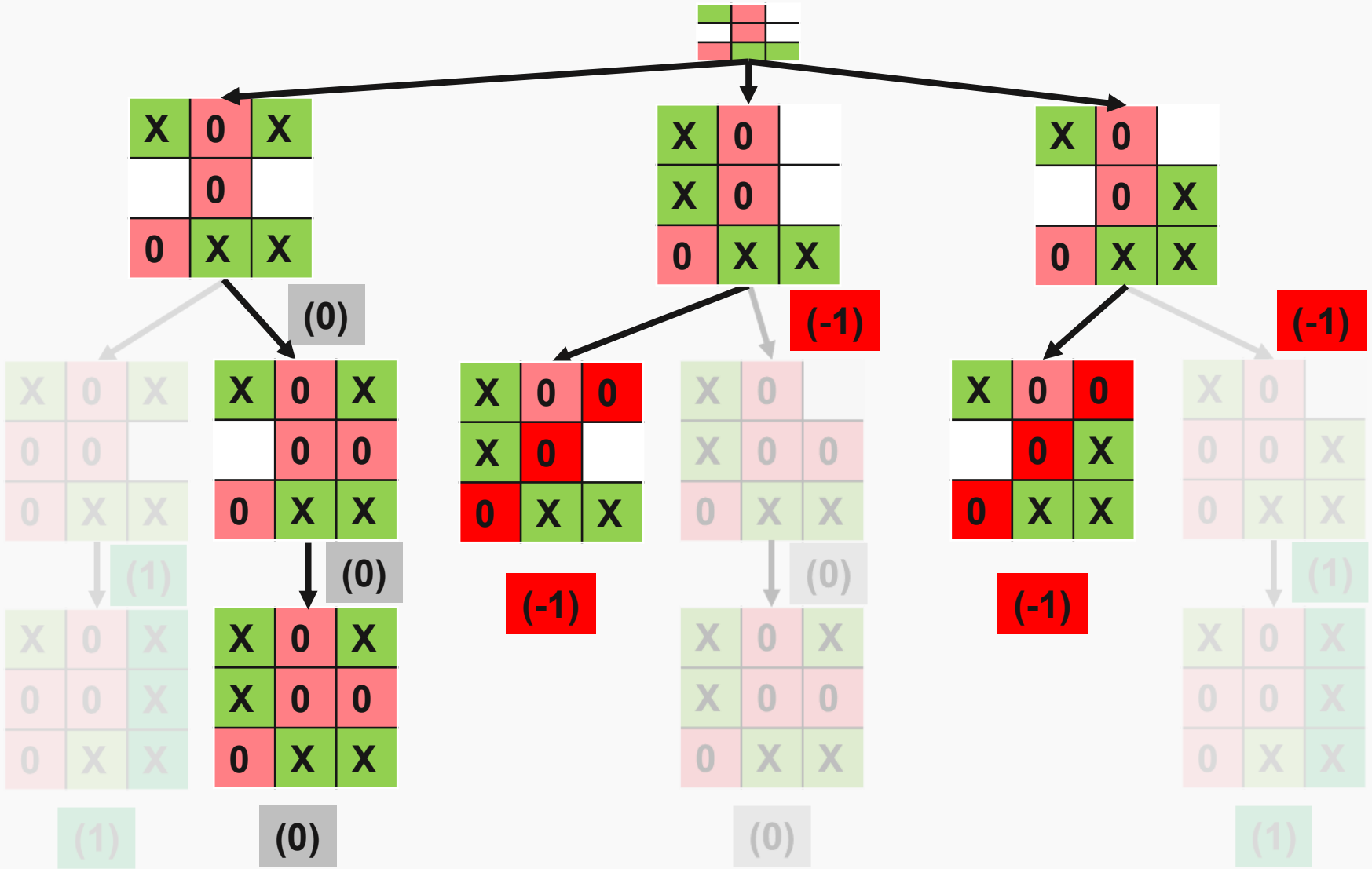
# MiniMax – X și 0



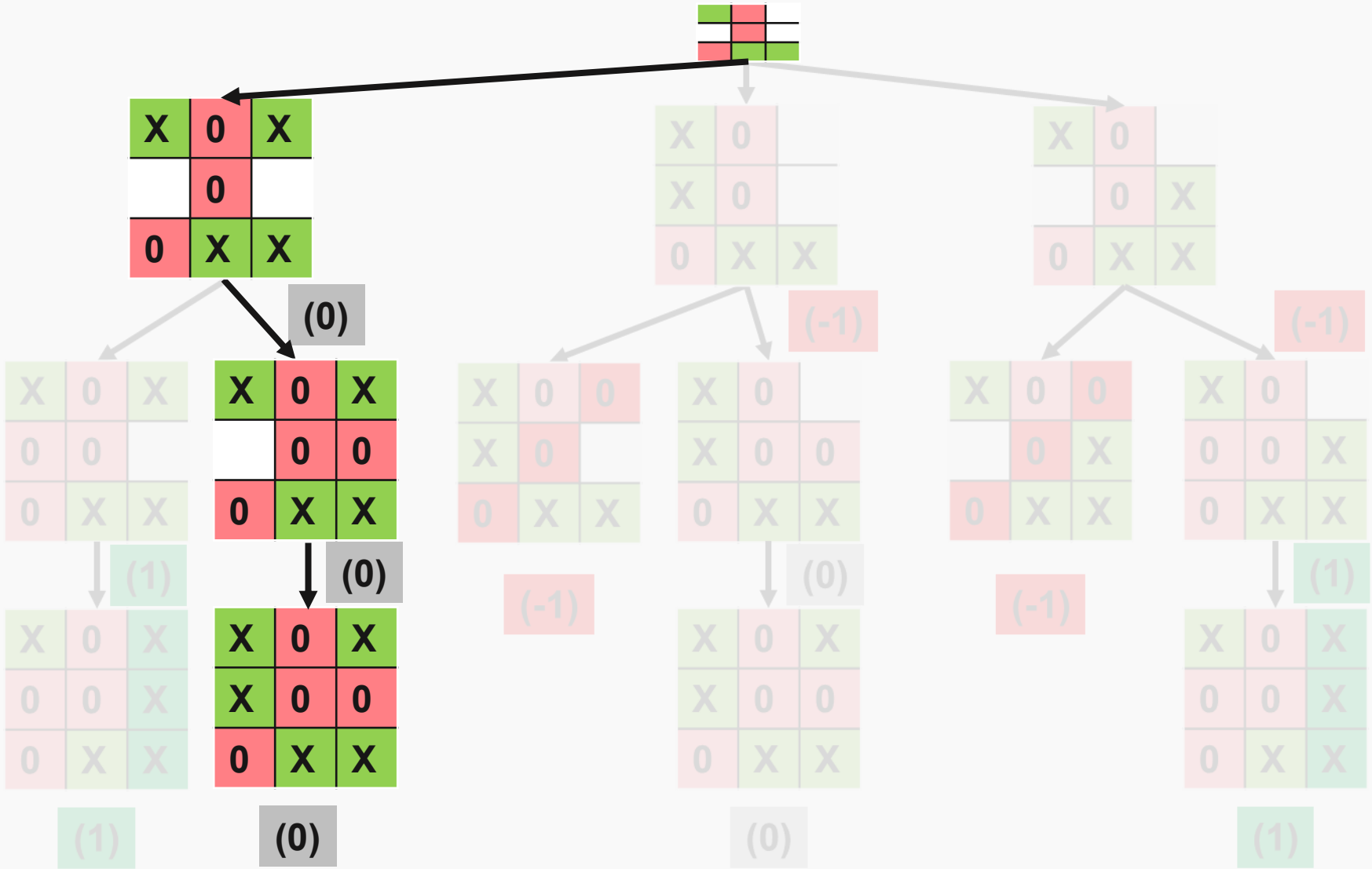
# MiniMax – X și 0



# MiniMax – X și 0



# MiniMax – X și 0



# MiniMax – Cât de mare este arborele?

- X și 0

  Aproape

  9*8*7*6*5*4*3*2*1 = 9! = 362,880 frunze

  sub 1M, deci încape în memorie.

- Dar șah?

# MiniMax – Cât de mare este arborele?

- În general nu putem analiza tot arborele. Este prea mare.

- Soluția: mergem până la o adâncime și încercăm să estimăm cât de avantajos sau dezavantajos este acea configurație a jocului.
- Estimăm -> Euristică (heuristic)

# MiniMax - Algoritm

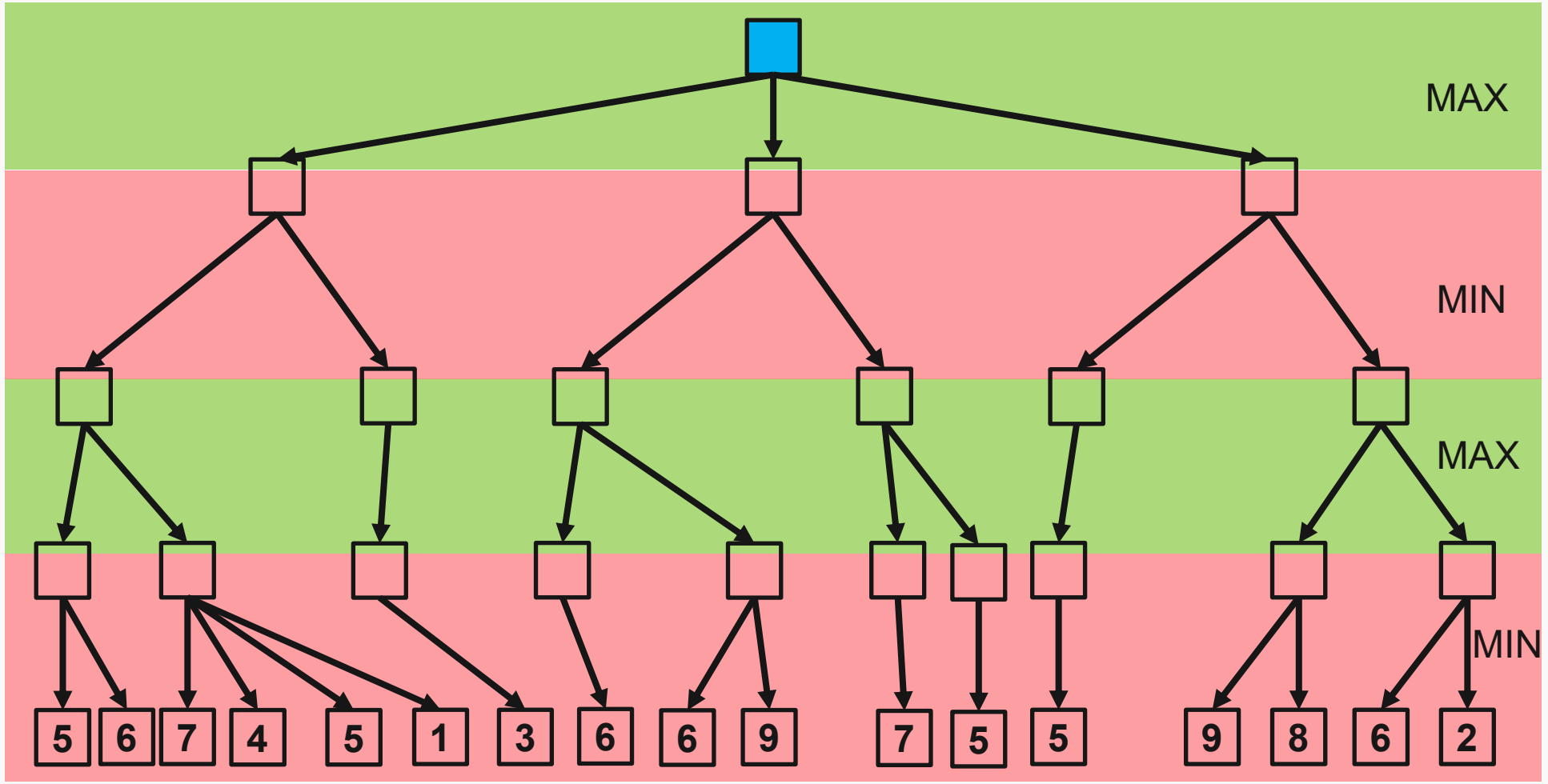
```
int miniMax(gameNode node, int level) {
  if (level == maxDepth || isEndGame(node))
        return heuristic(node);

  if (level % 2 == maximizingPlayer) {
    int value = −∞;
    for each (child of node)
      value = max(value, miniMax(child, level + 1));
    return value;
  } else {
    int value = +∞;
    for each (child of node)
      value = min(value, miniMax(child, level + 1));
    return value;
  }
}
```

# Apelare MiniMax

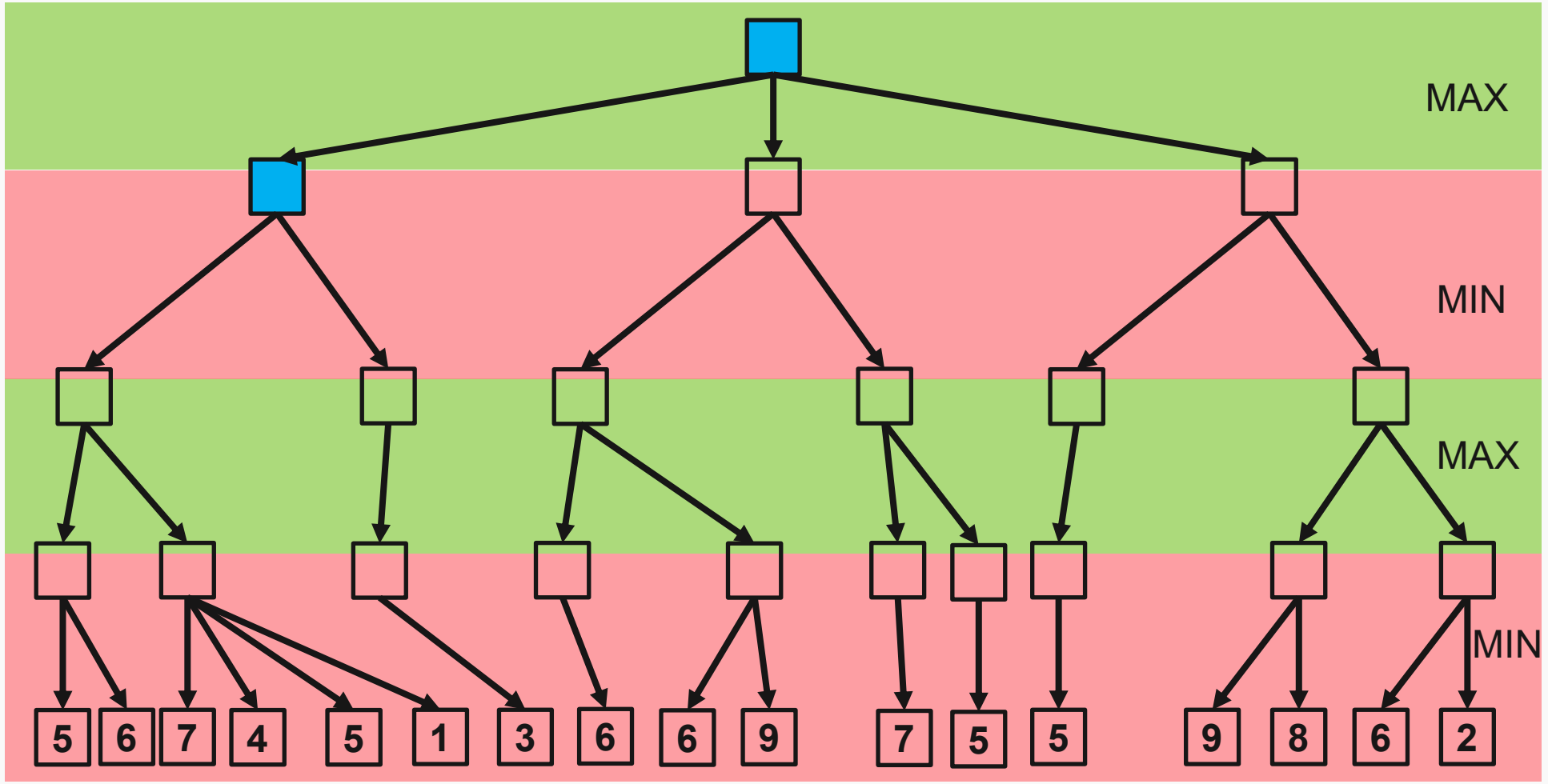
```
for each (child of node)
   score[child] = miniMax(child, 0);
```

Mutarea este acel child pentru care score e maxim

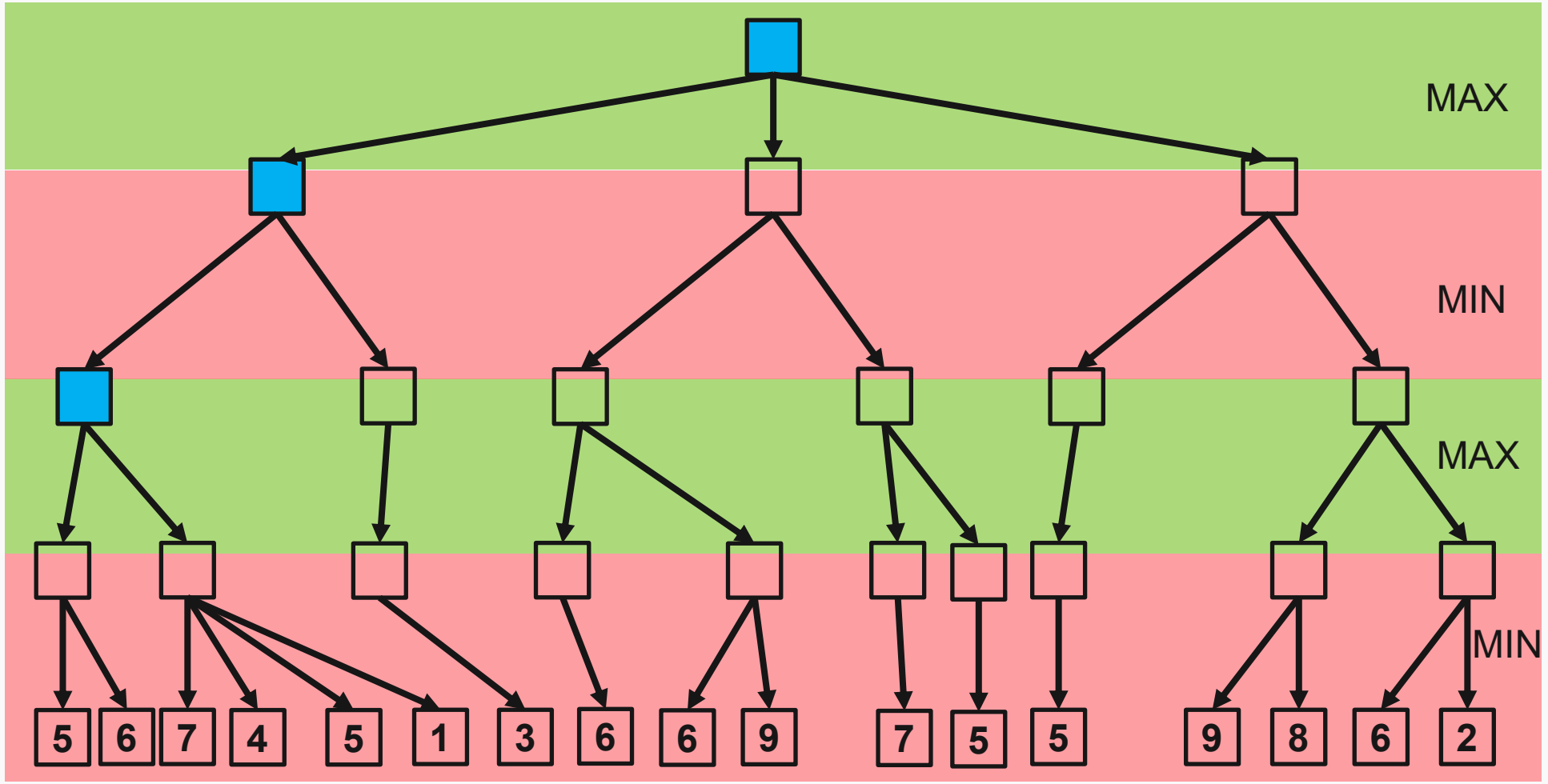
# AlphaBeta Prunning Ideea



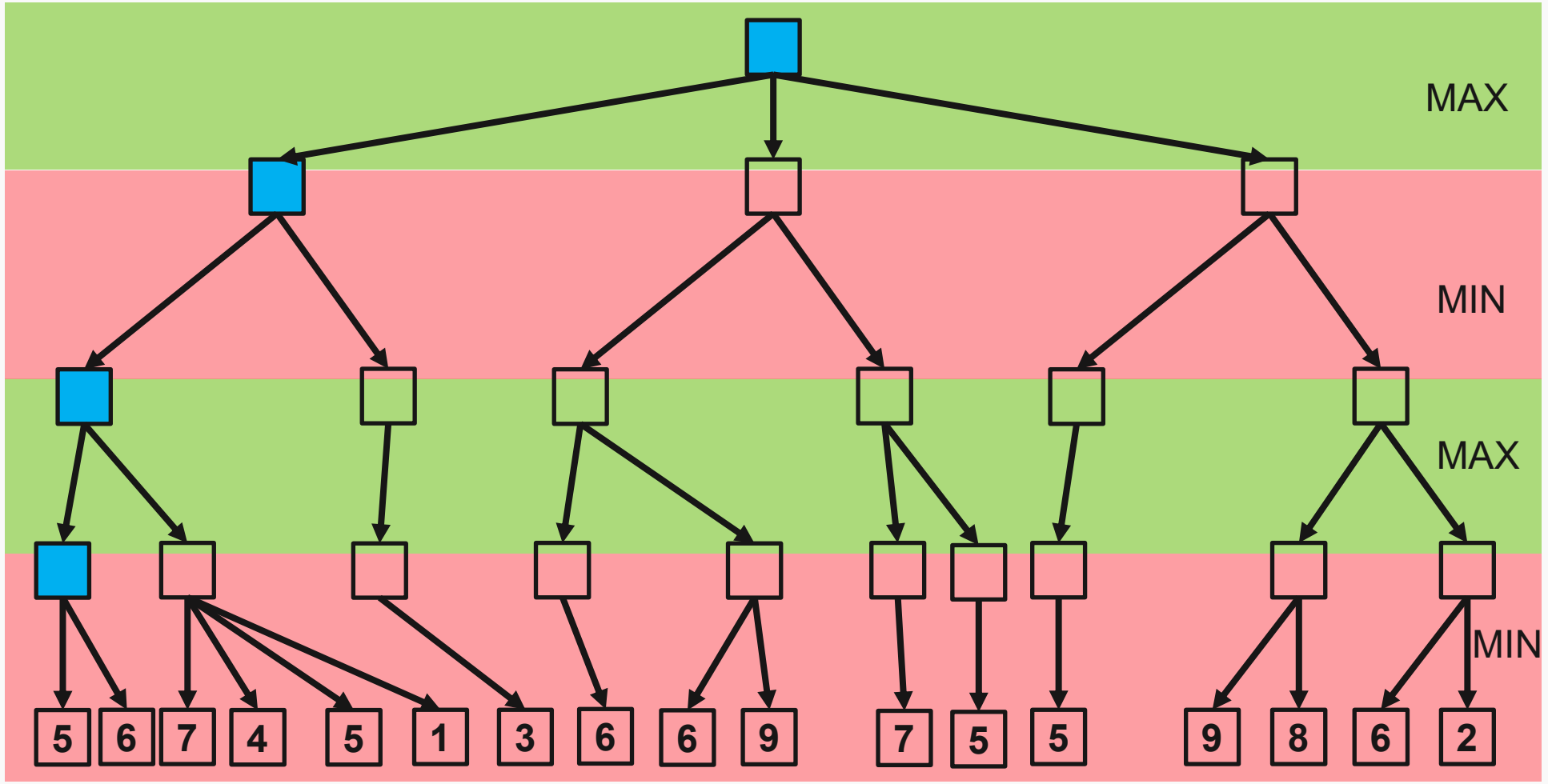
# AlphaBeta Prunning Ideea



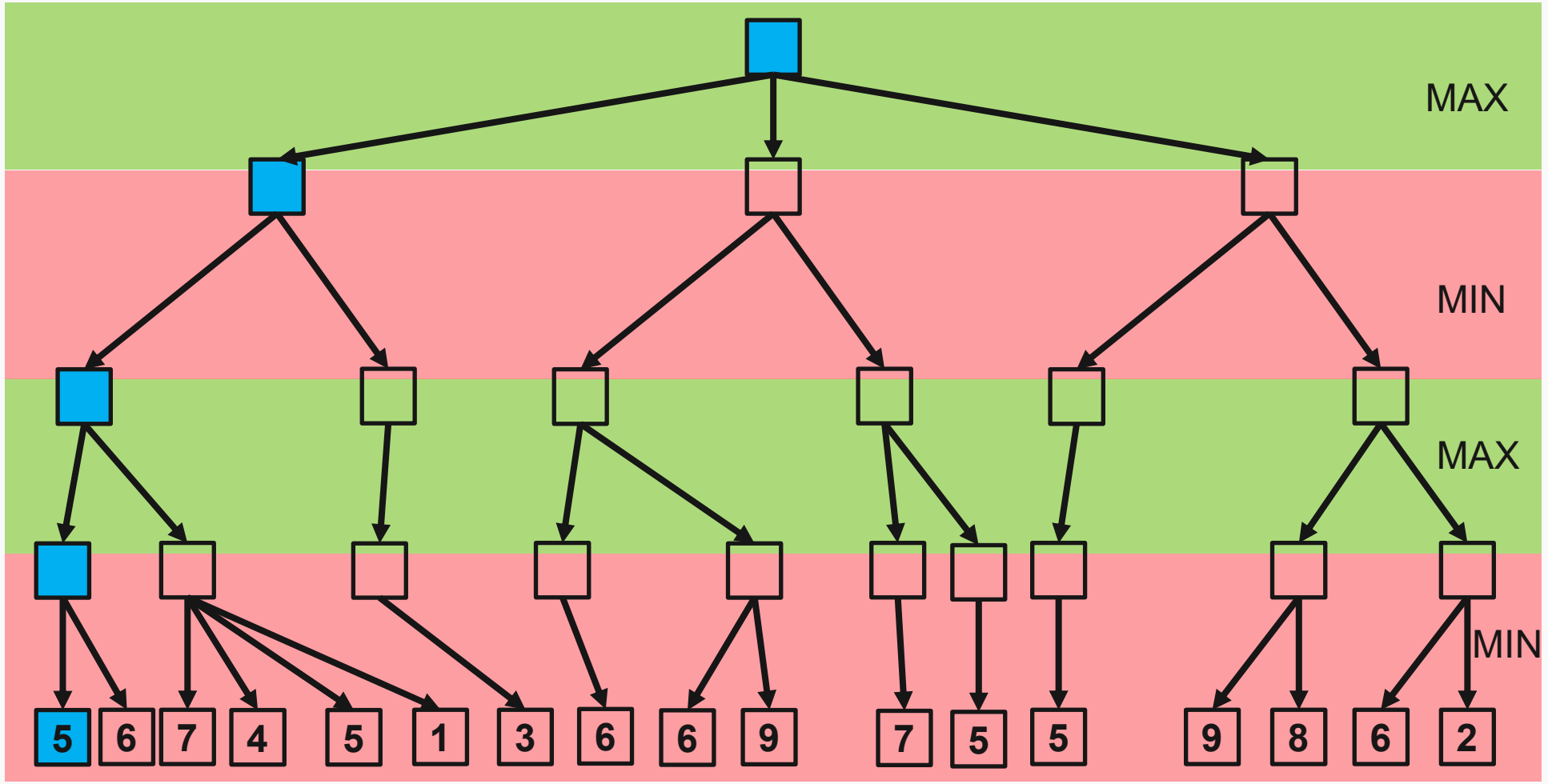
# AlphaBeta Prunning Ideea



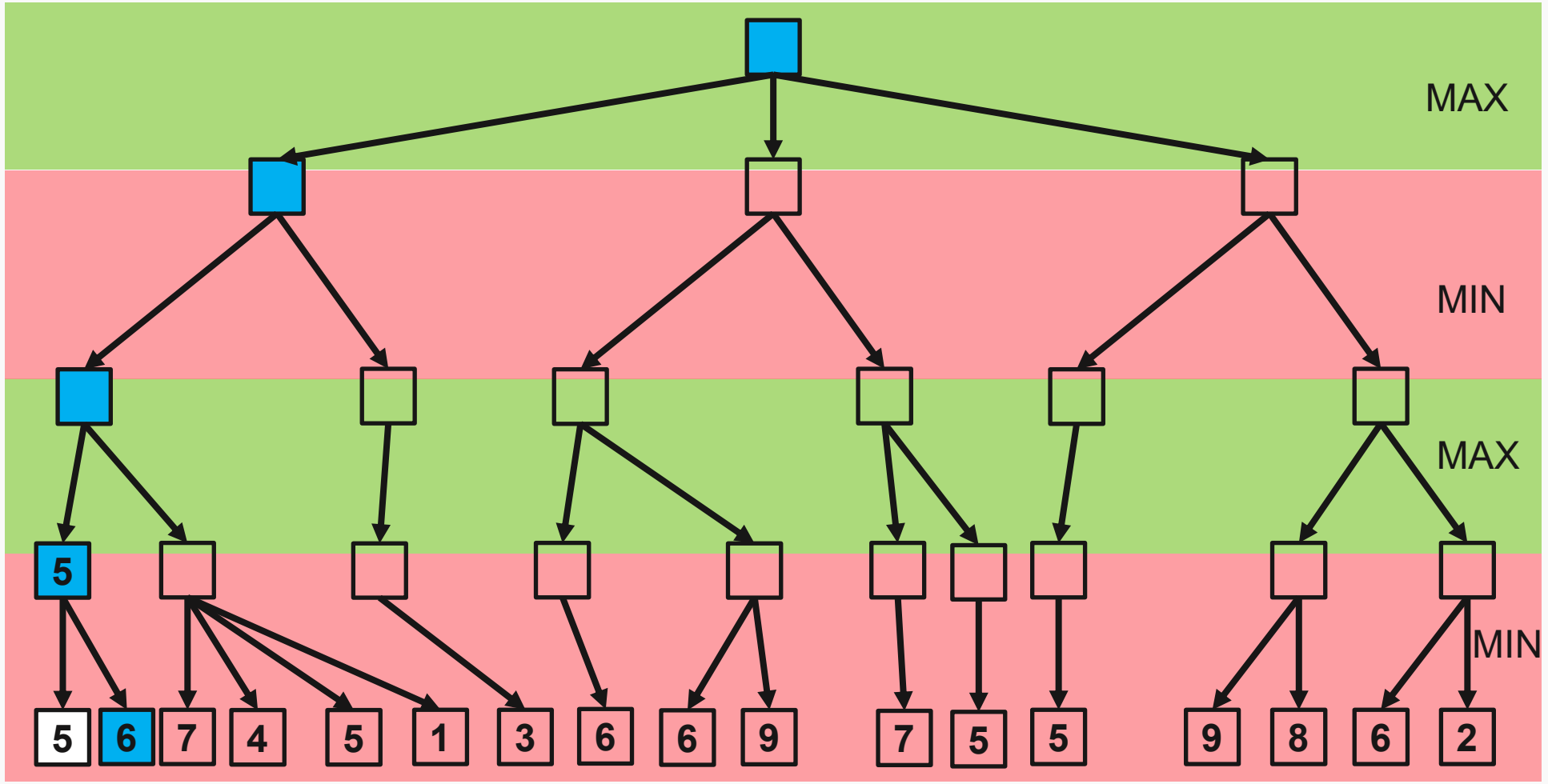
# AlphaBeta Prunning Ideea



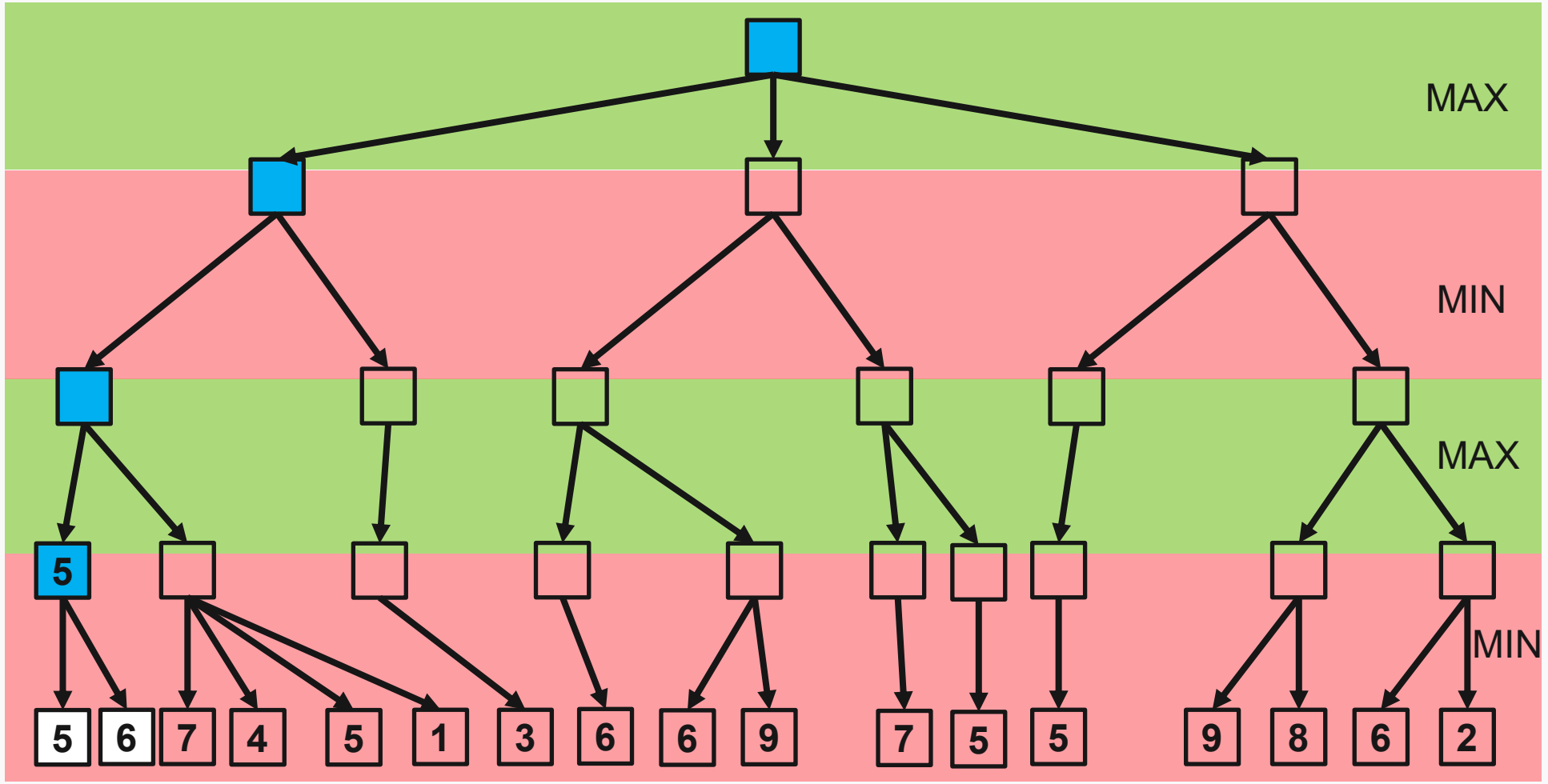
# AlphaBeta Prunning Ideea



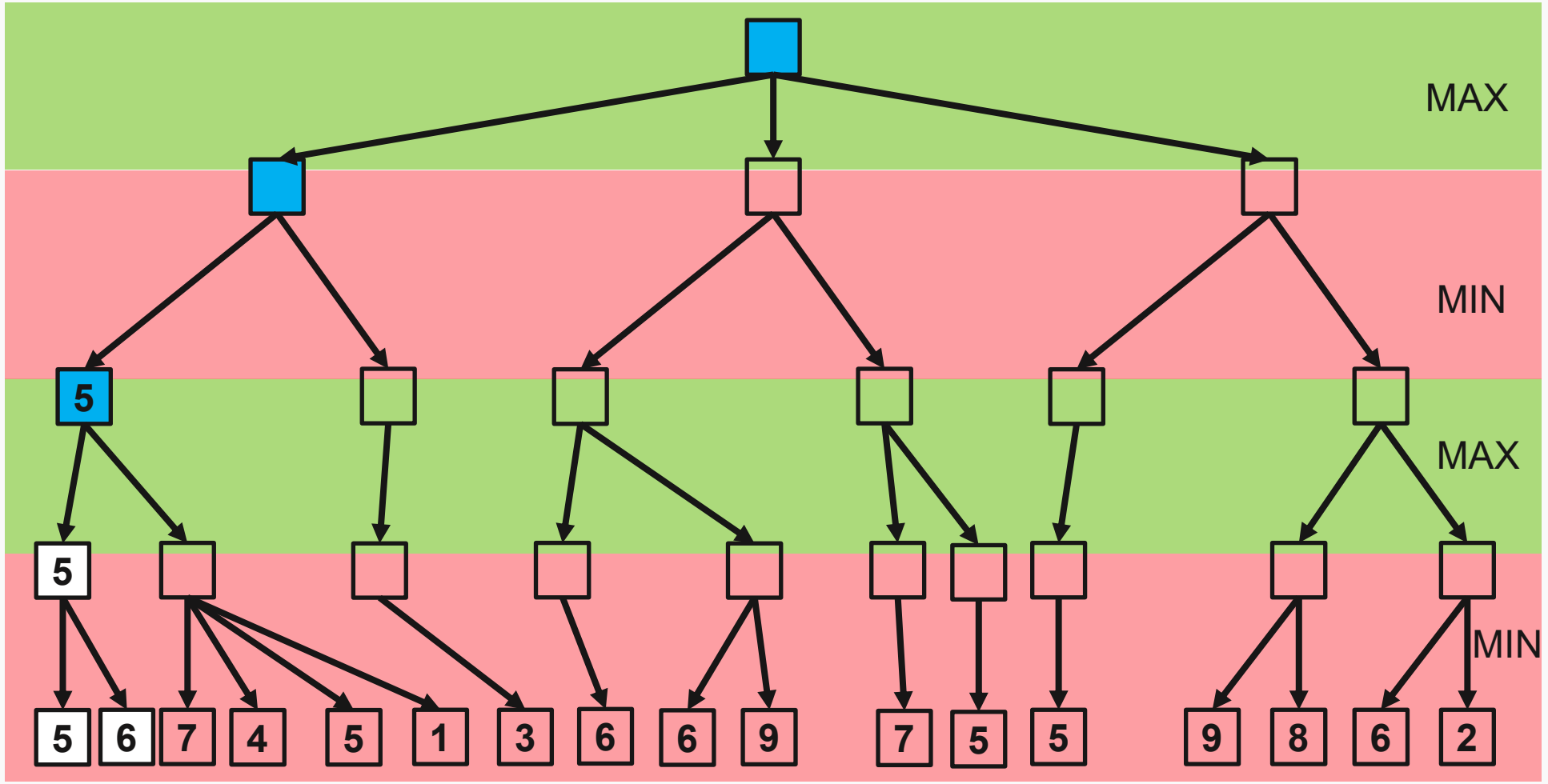
# AlphaBeta Prunning Ideea



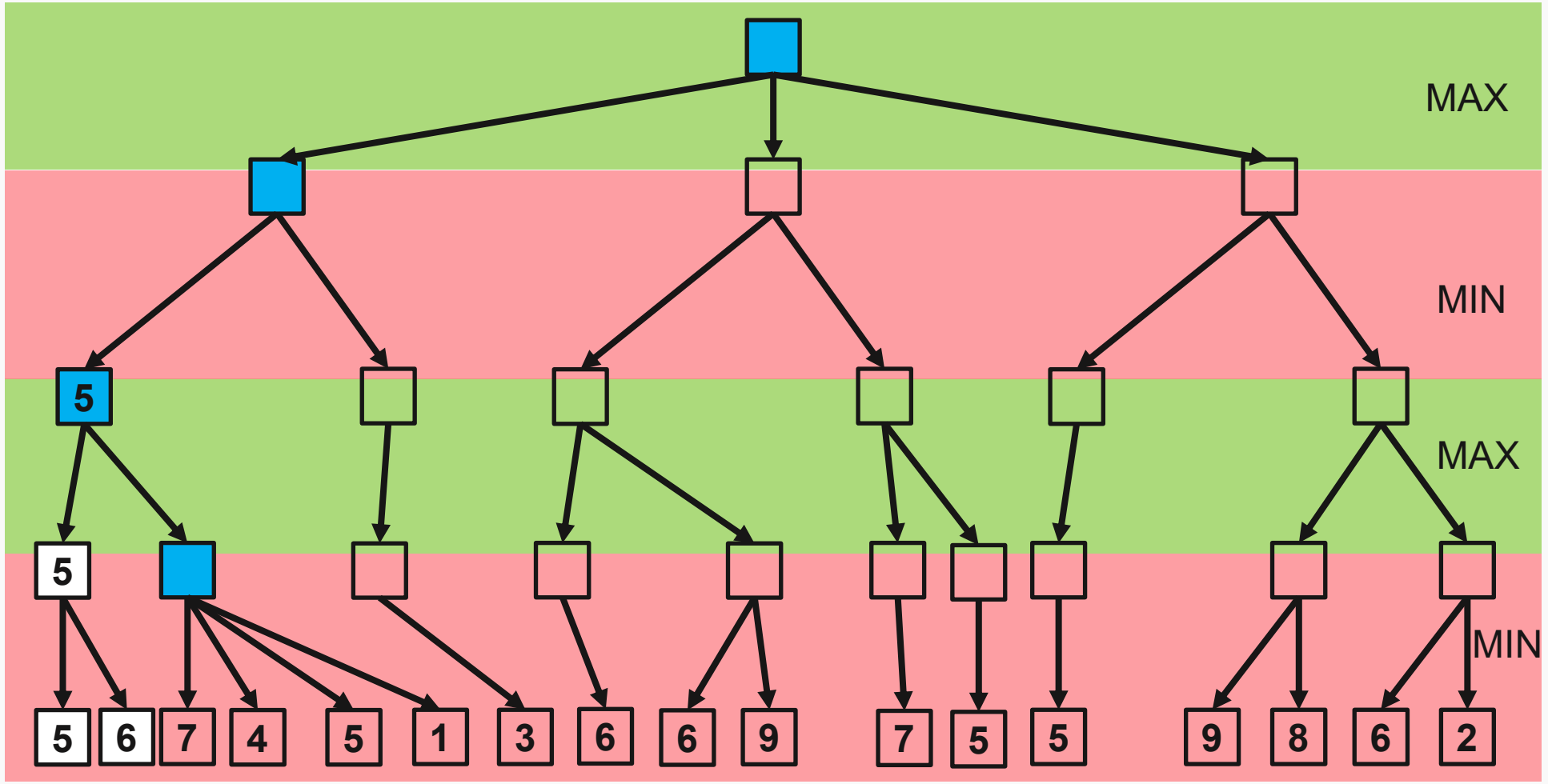
# AlphaBeta Prunning Ideea



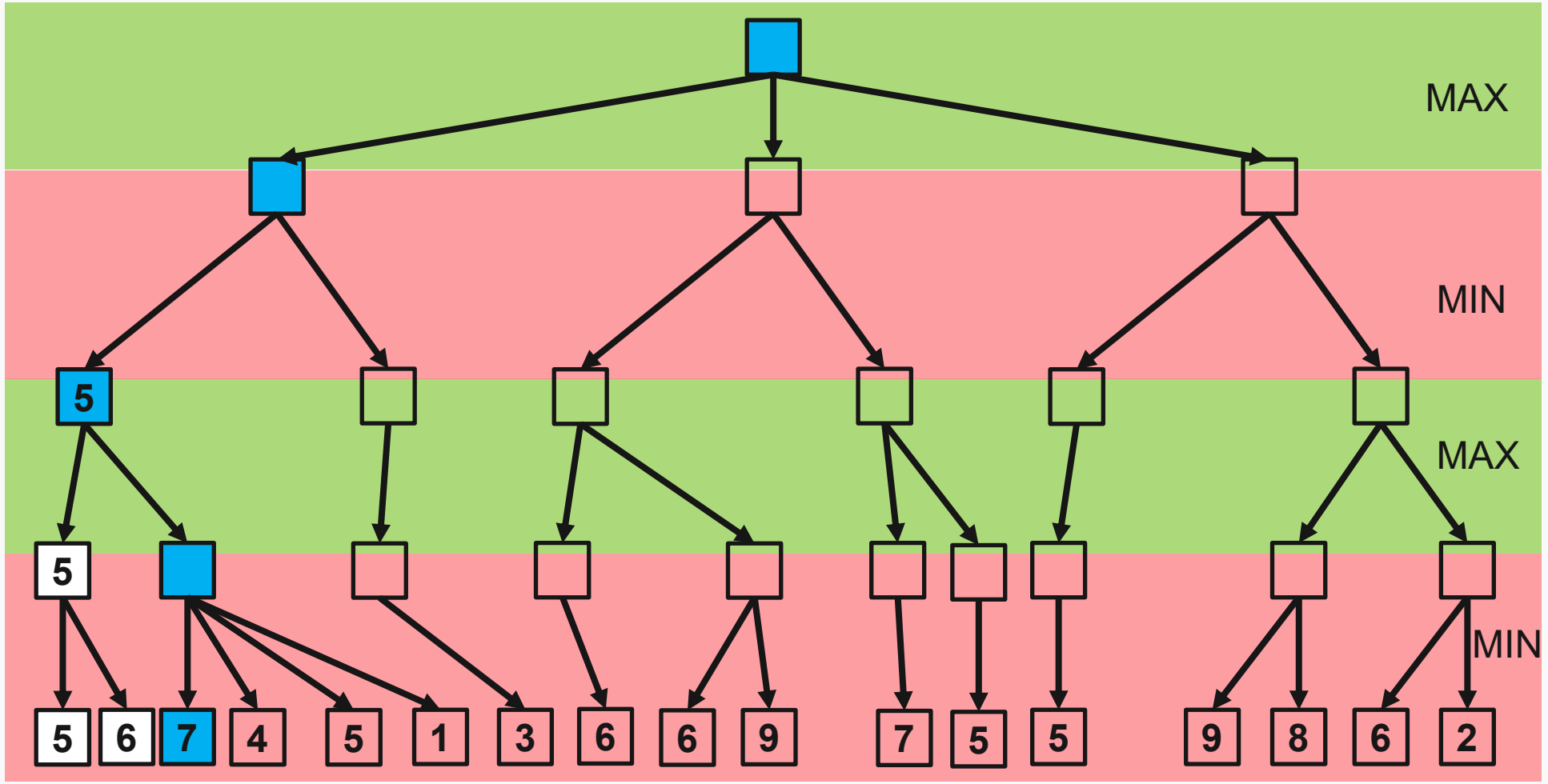
# AlphaBeta Prunning Ideea



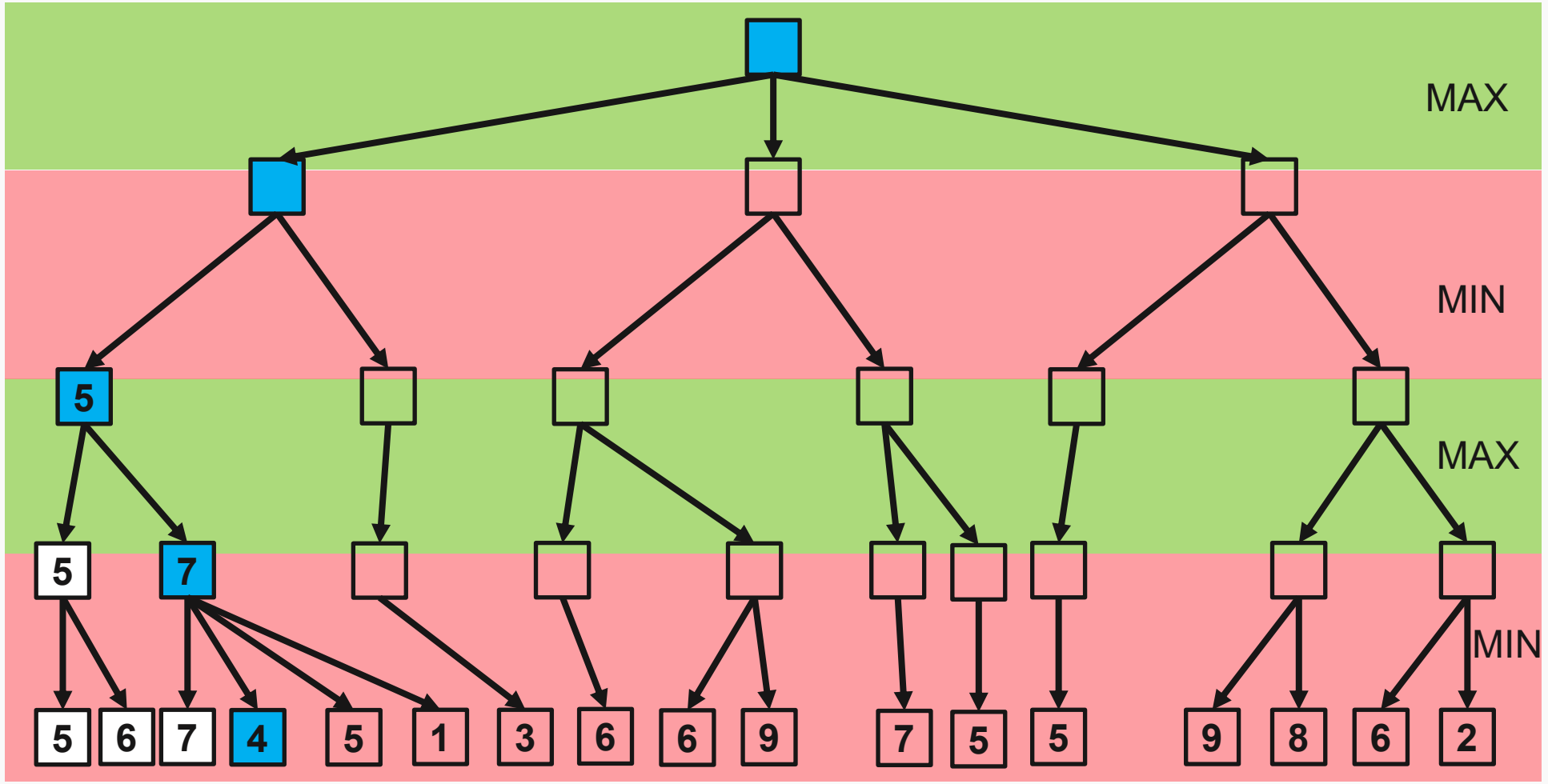
# AlphaBeta Prunning Ideea



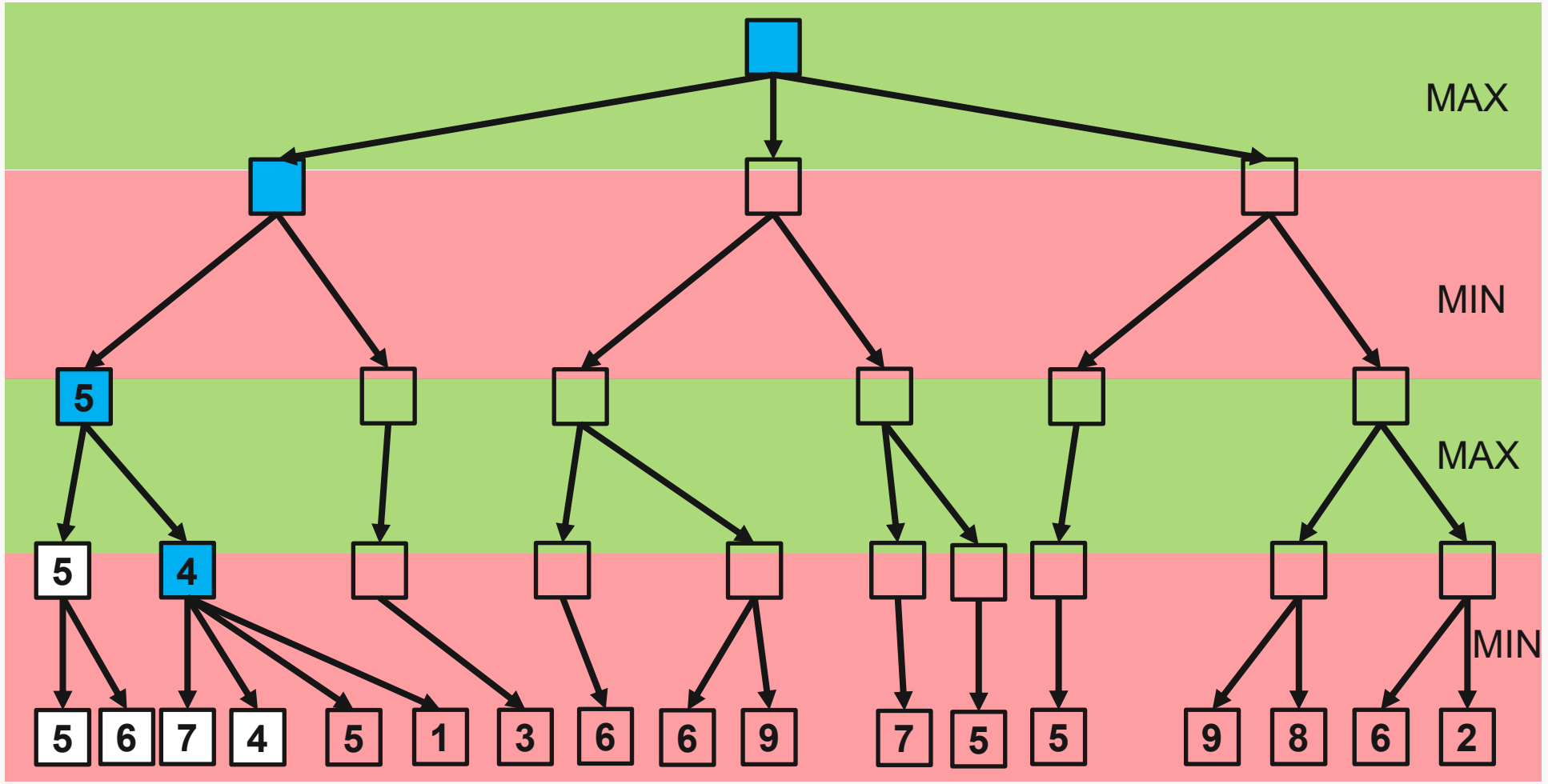
# AlphaBeta Prunning Ideea



# AlphaBeta Prunning Ideea

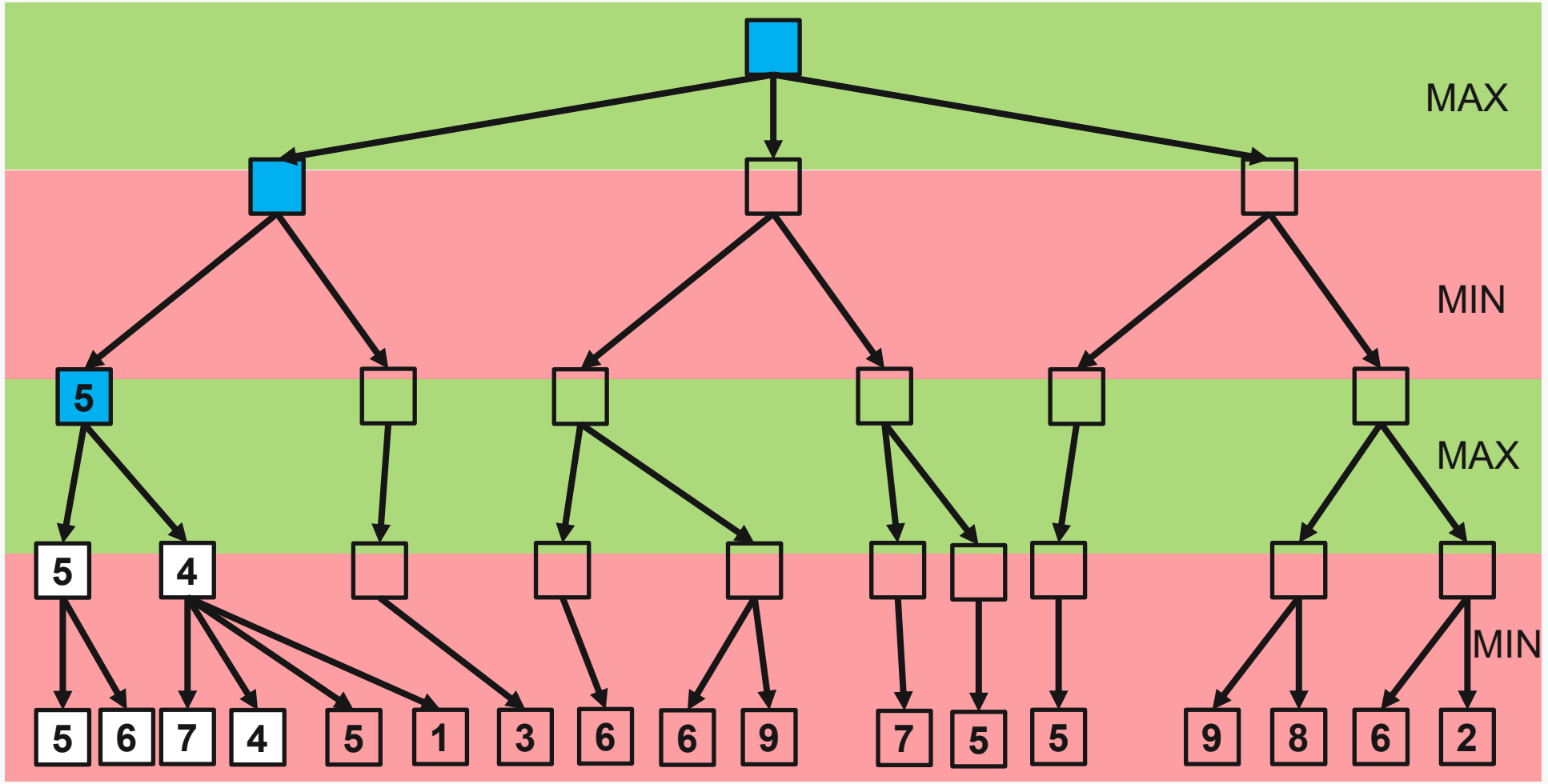


# AlphaBeta Prunning Ideea

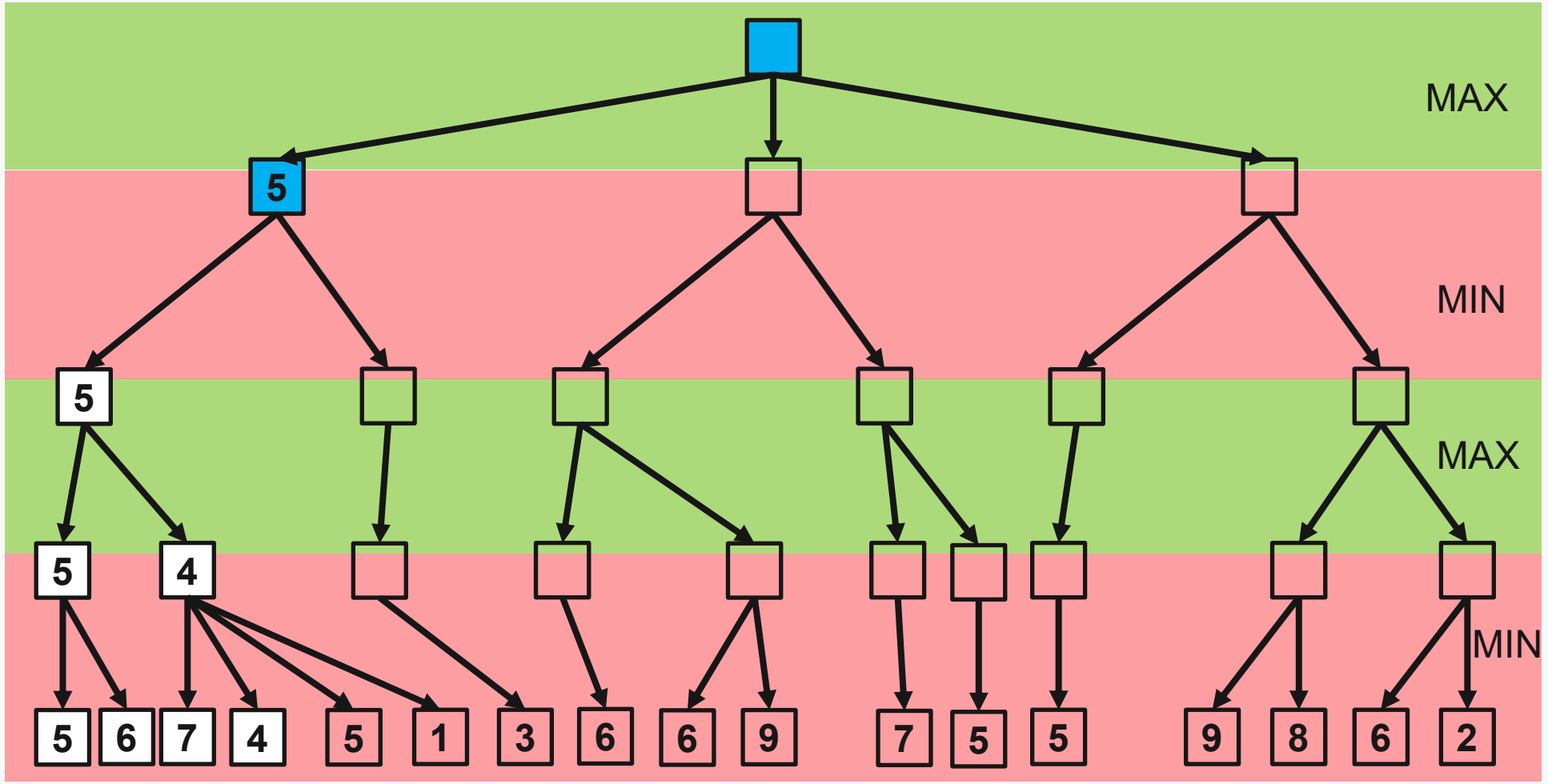



**Suntem deja siguri că la acest nivel competitorul va alege 4 sau mai mic.**
**Dar dacă el alege 4 sau mai mic, noi știm siguri că vom alege 5.**

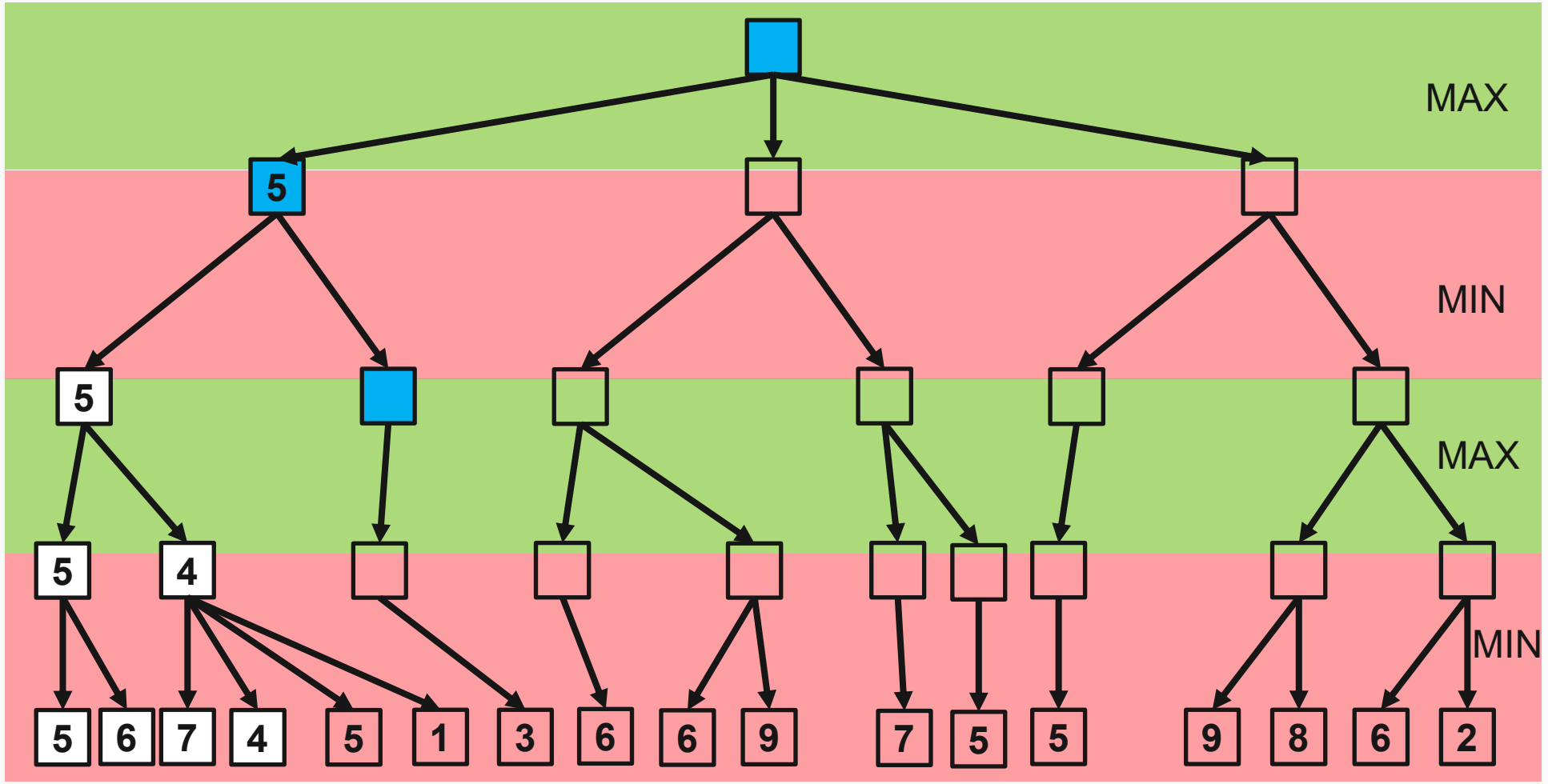
# AlphaBeta Prunning Ideea



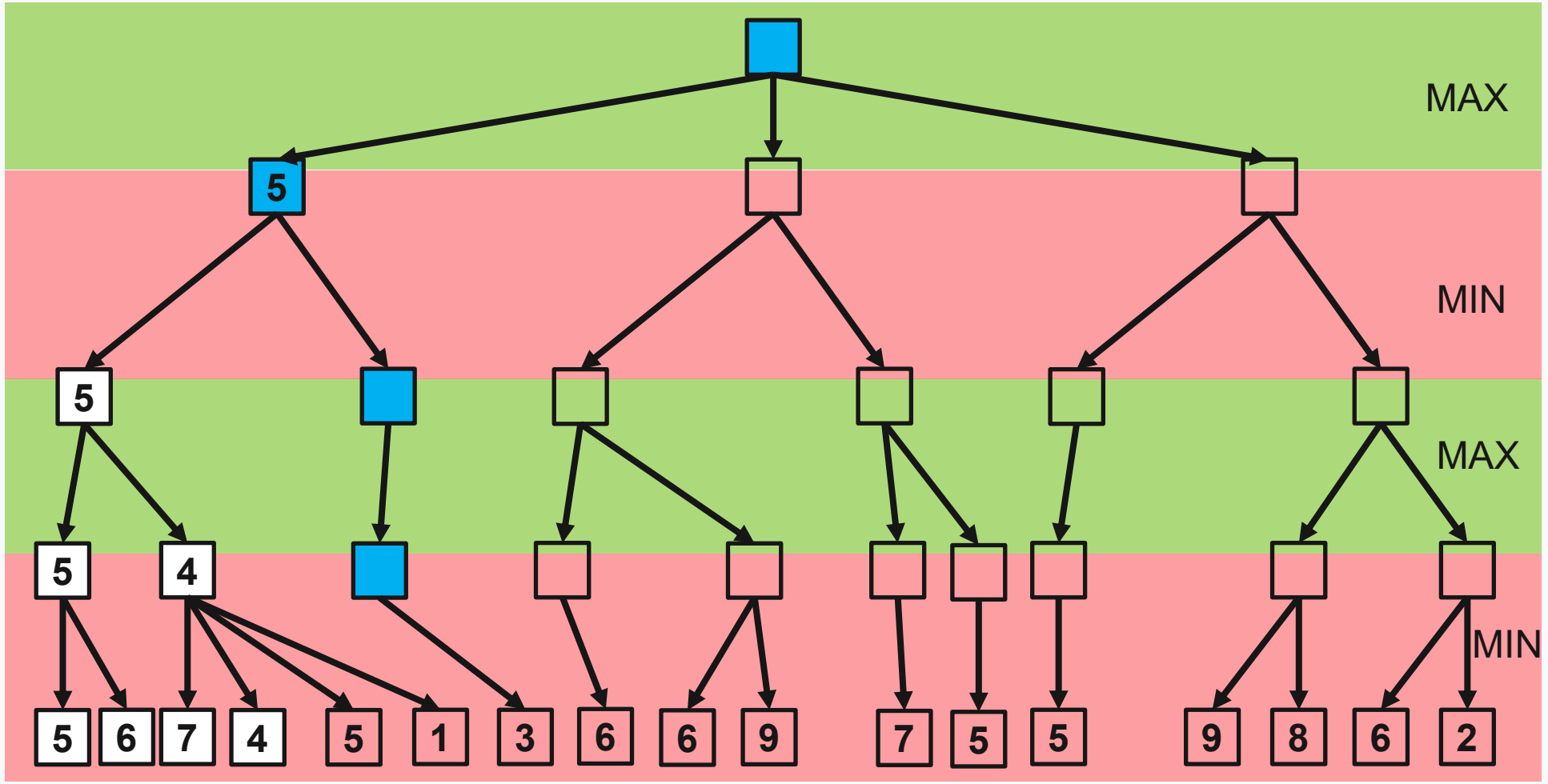
# AlphaBeta Prunning Ideea



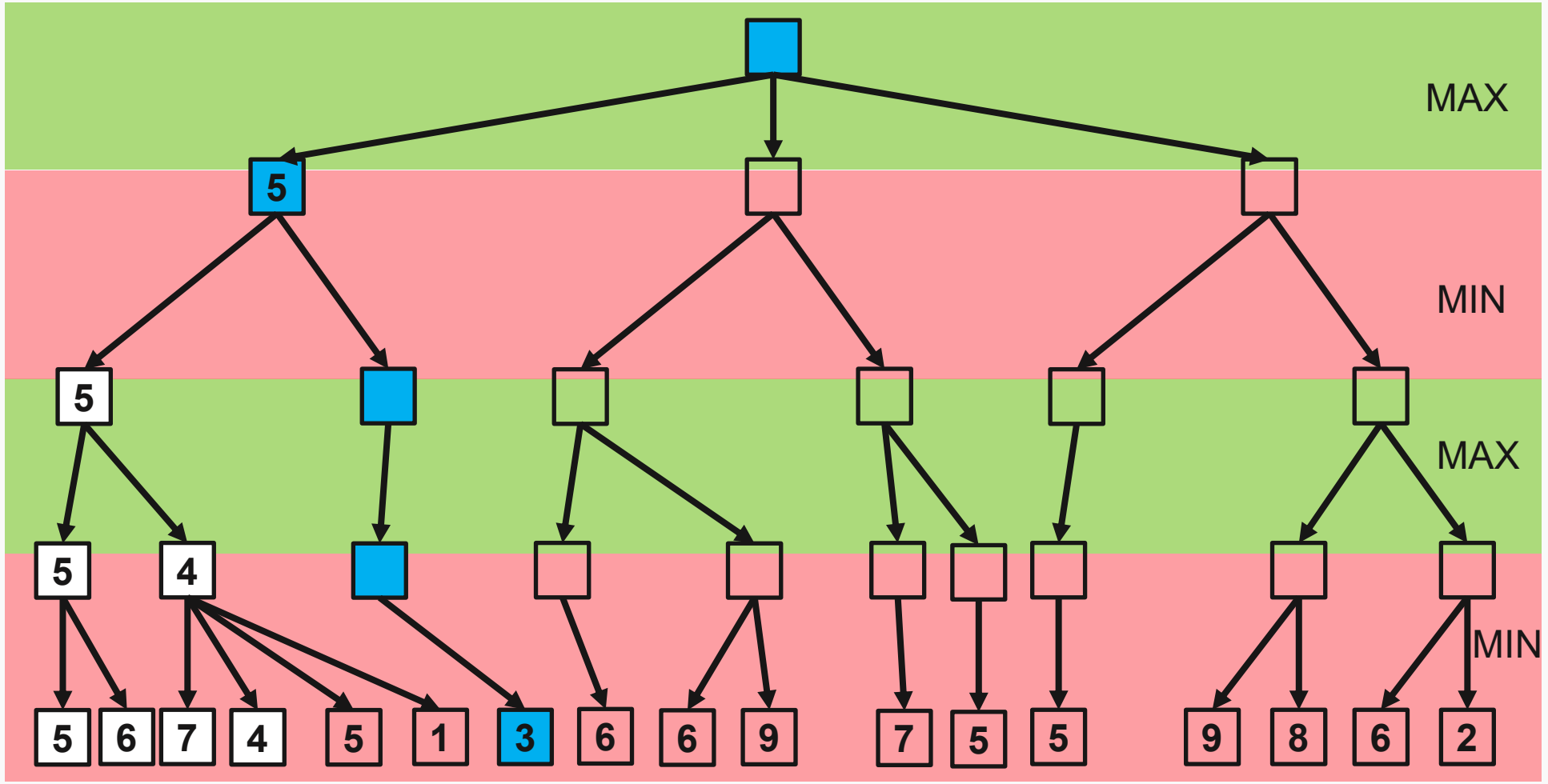
# AlphaBeta Prunning Ideea



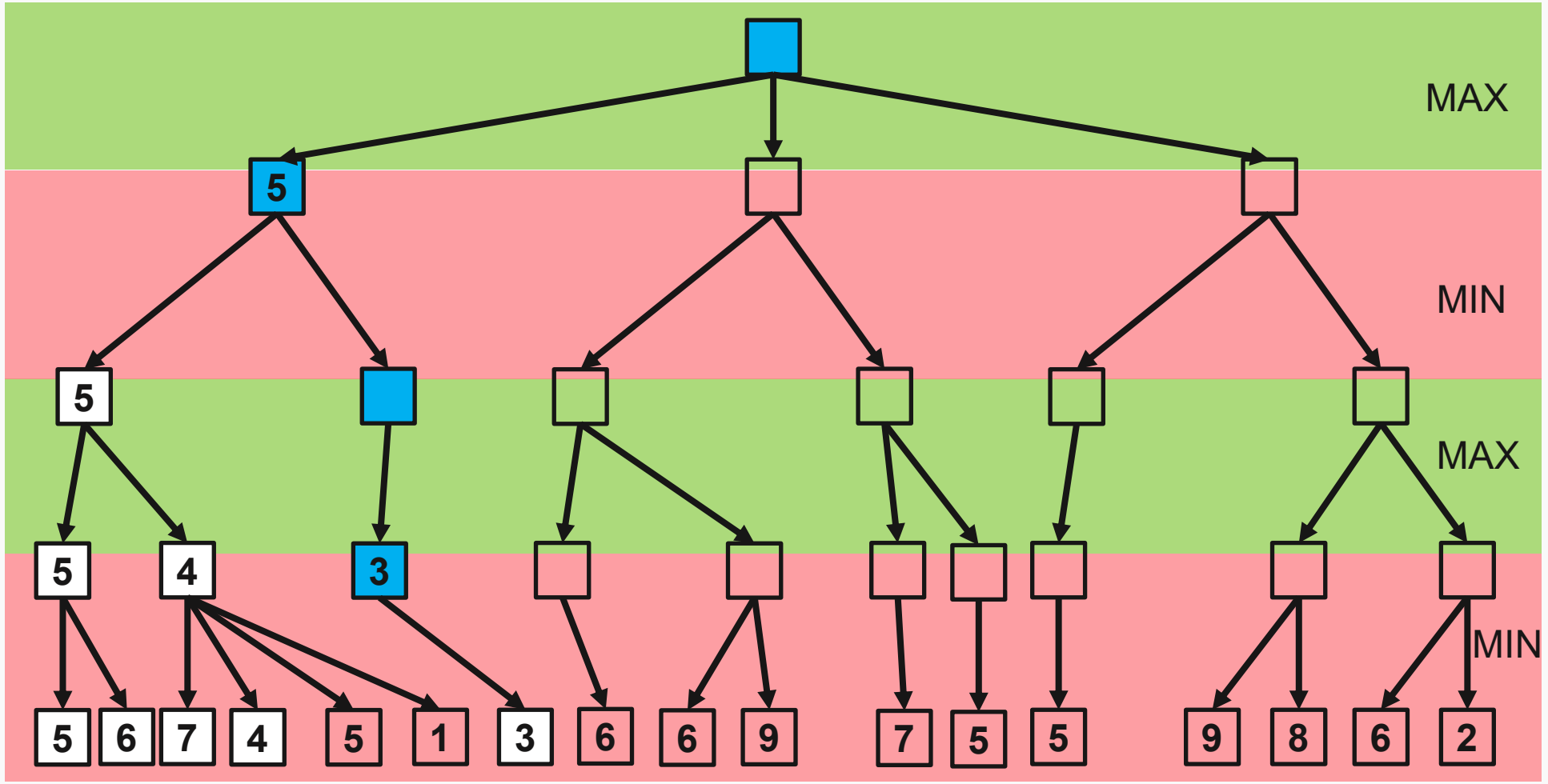
# AlphaBeta Prunning Ideea



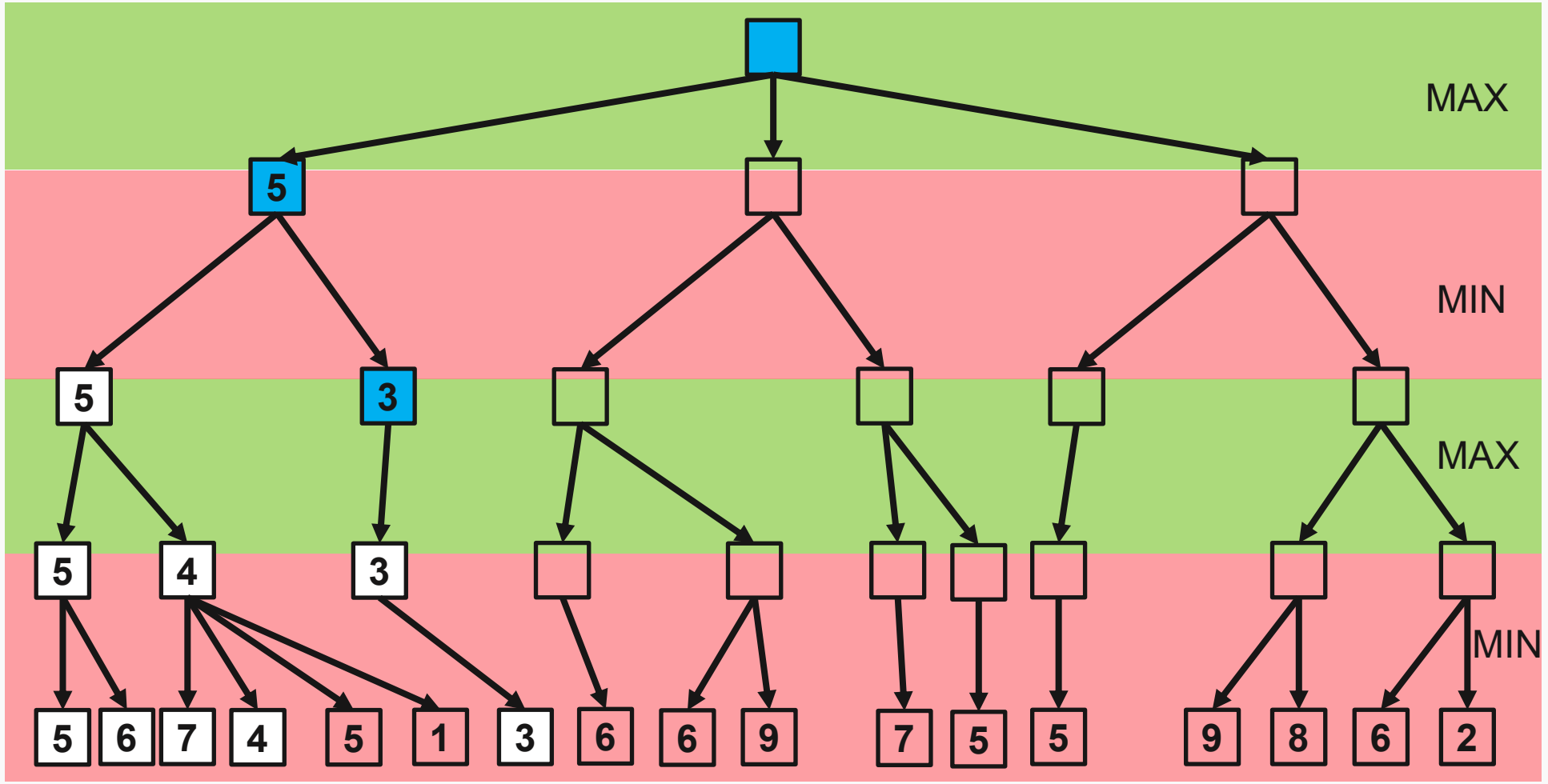
# AlphaBeta Prunning Ideea



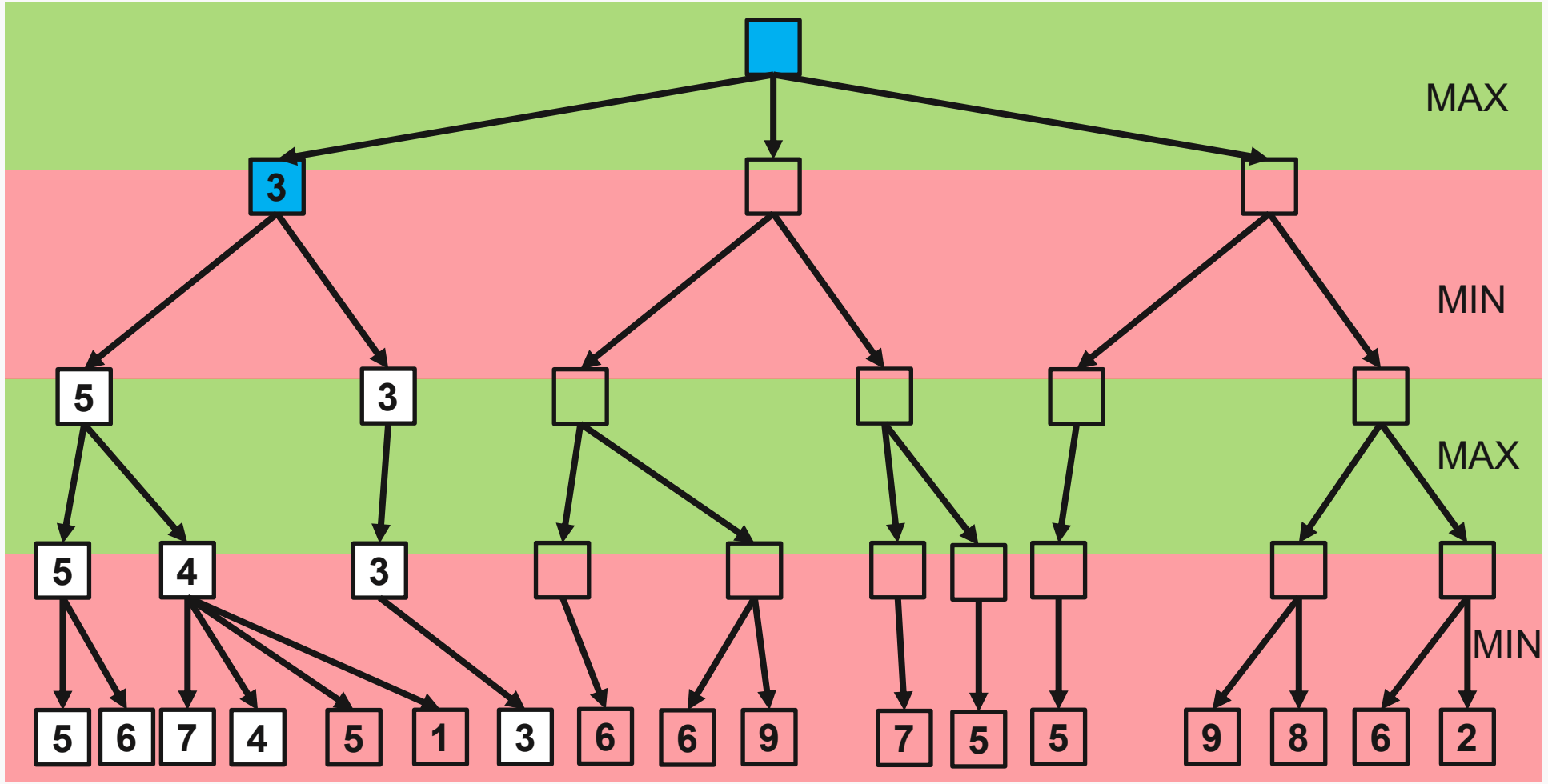
# AlphaBeta Prunning Ideea



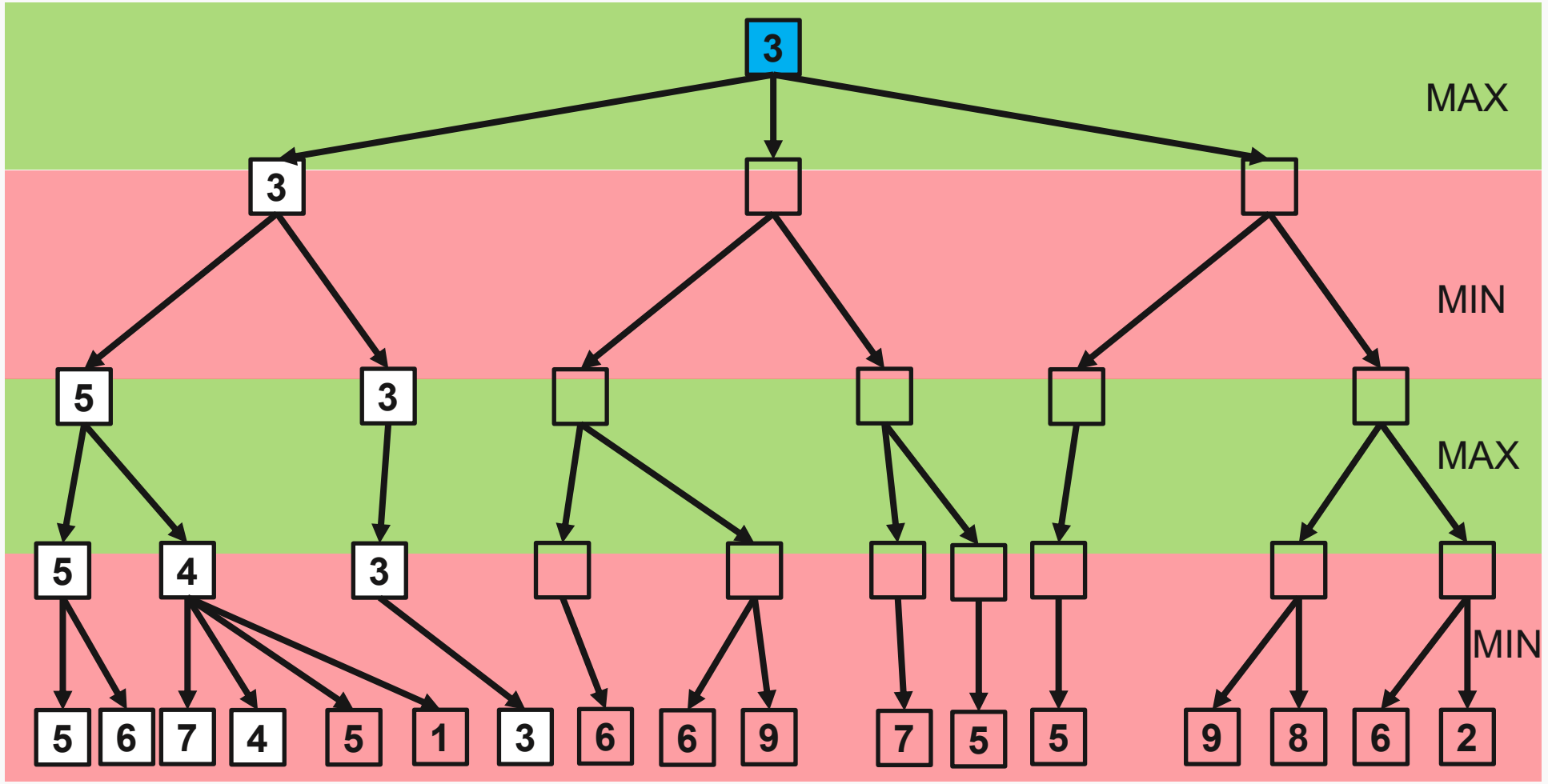
# AlphaBeta Prunning Ideea



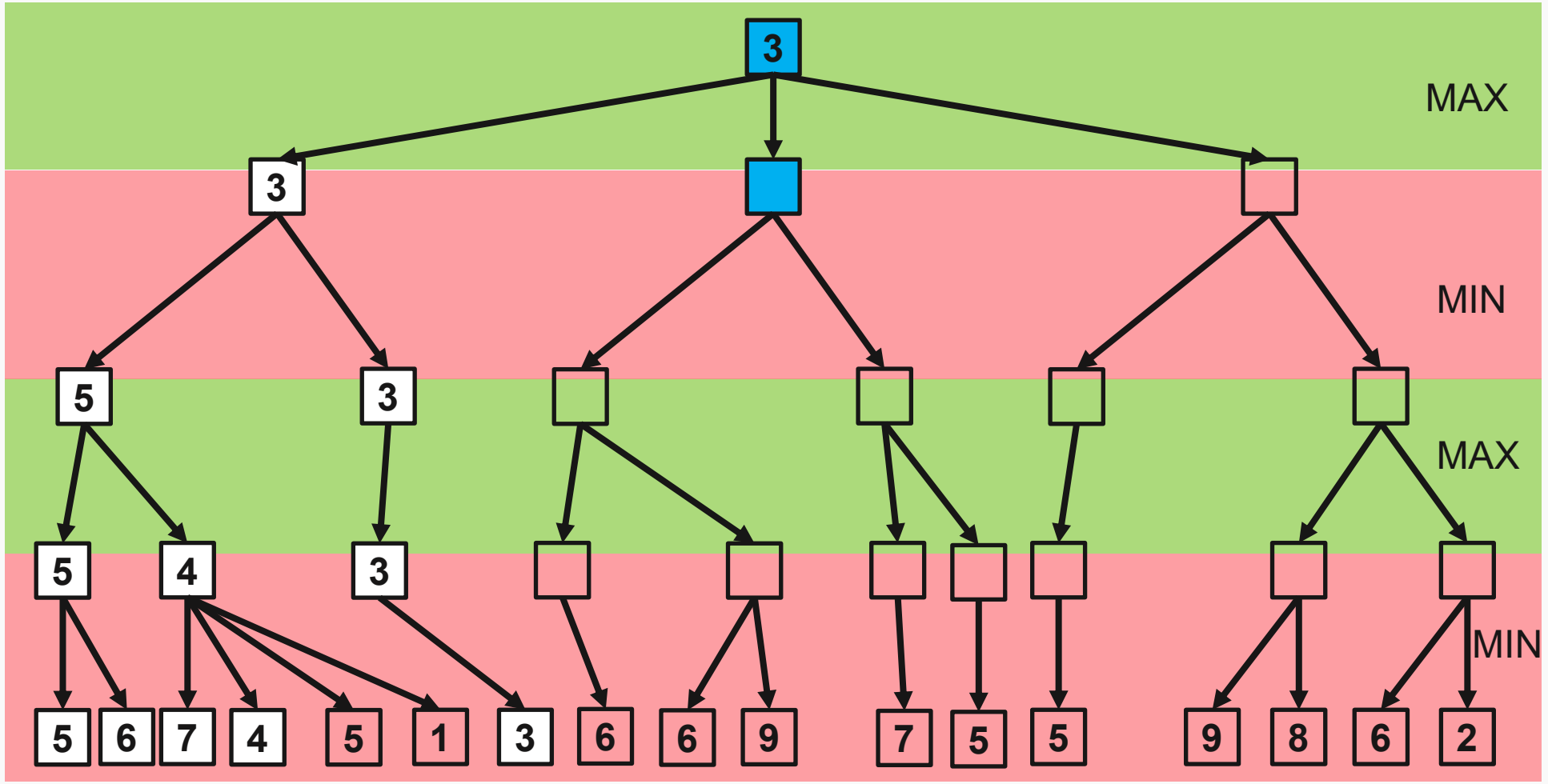
# AlphaBeta Prunning Ideea



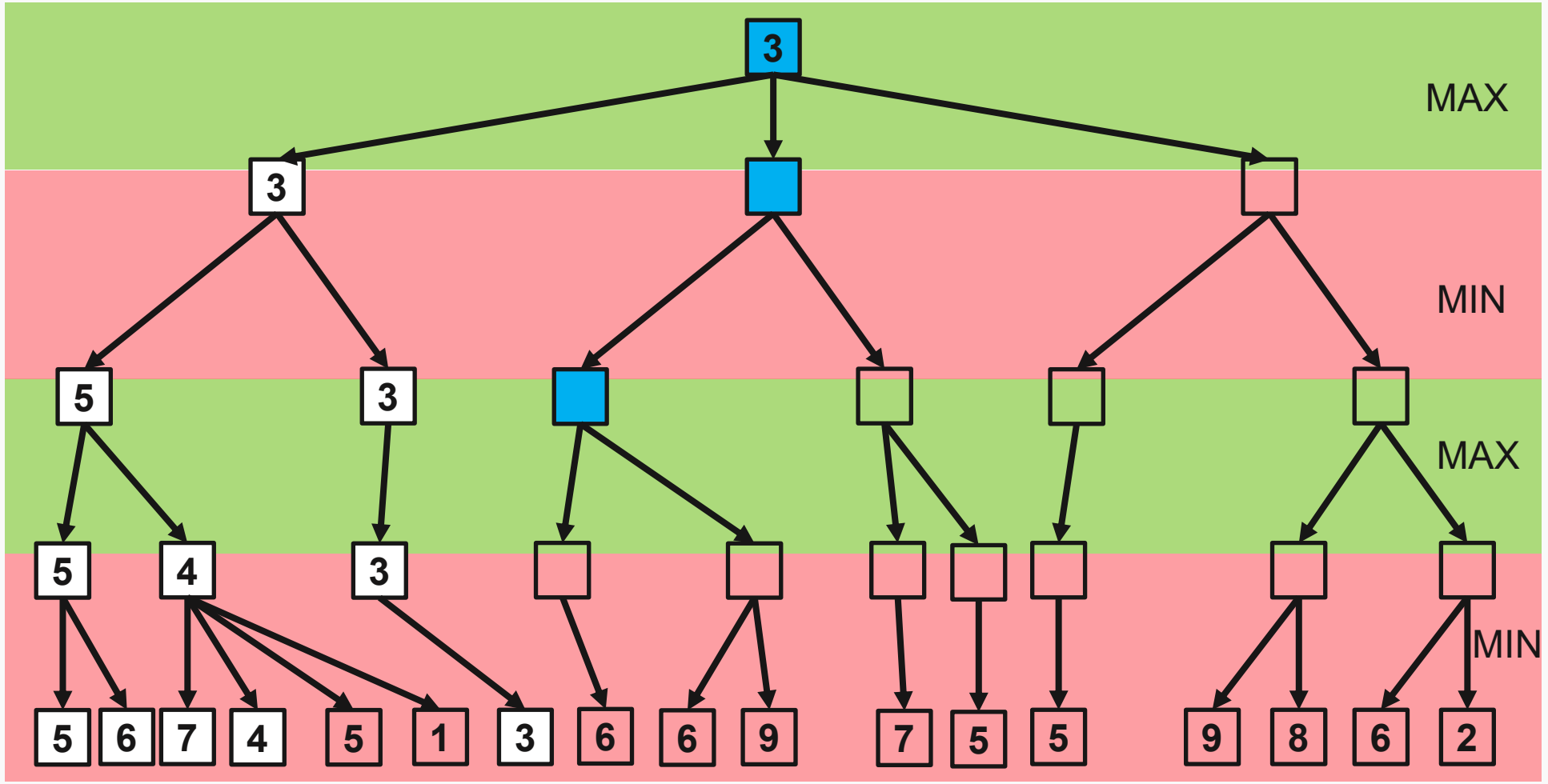
# AlphaBeta Prunning Ideea



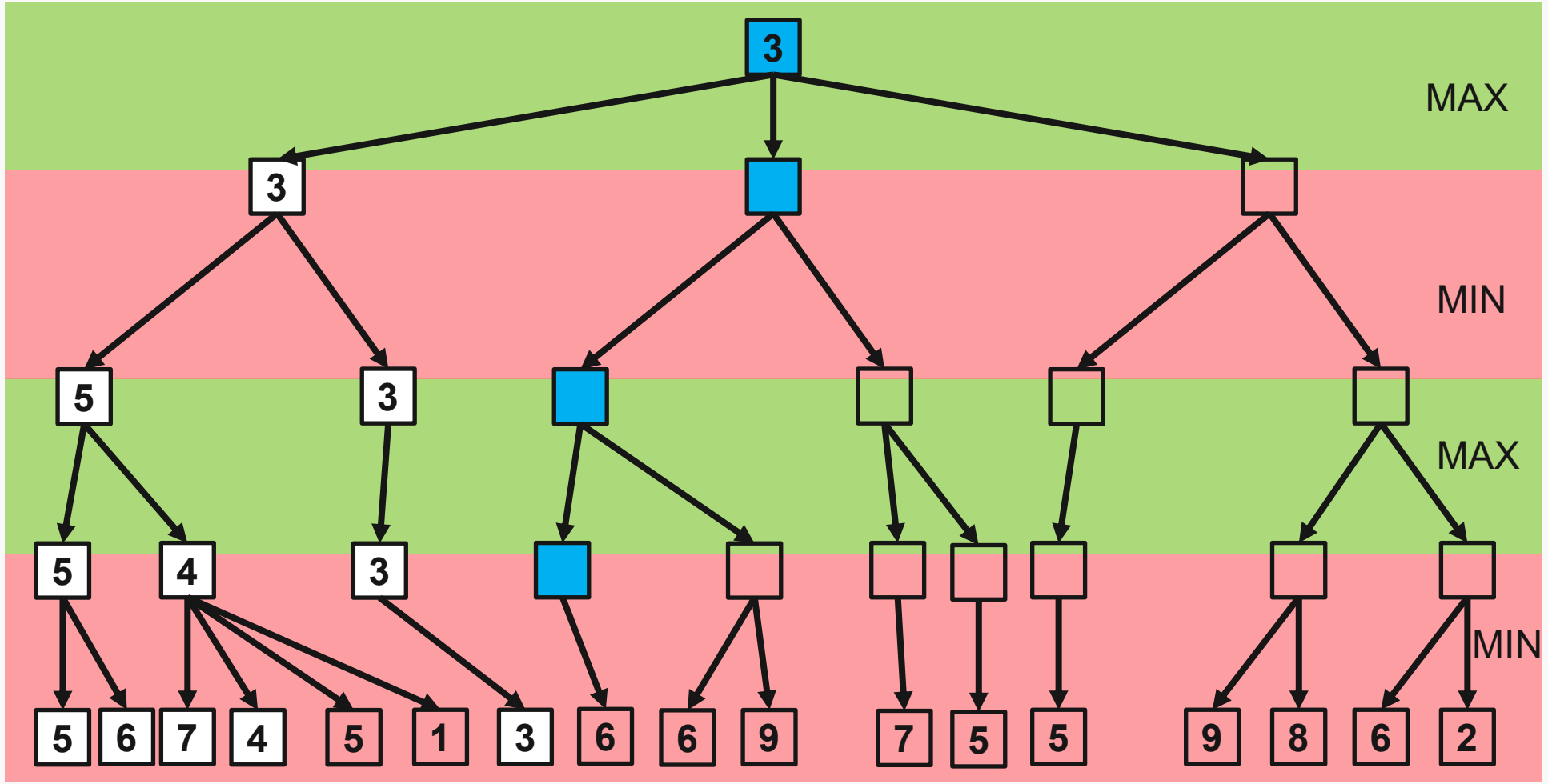
# AlphaBeta Prunning Ideea



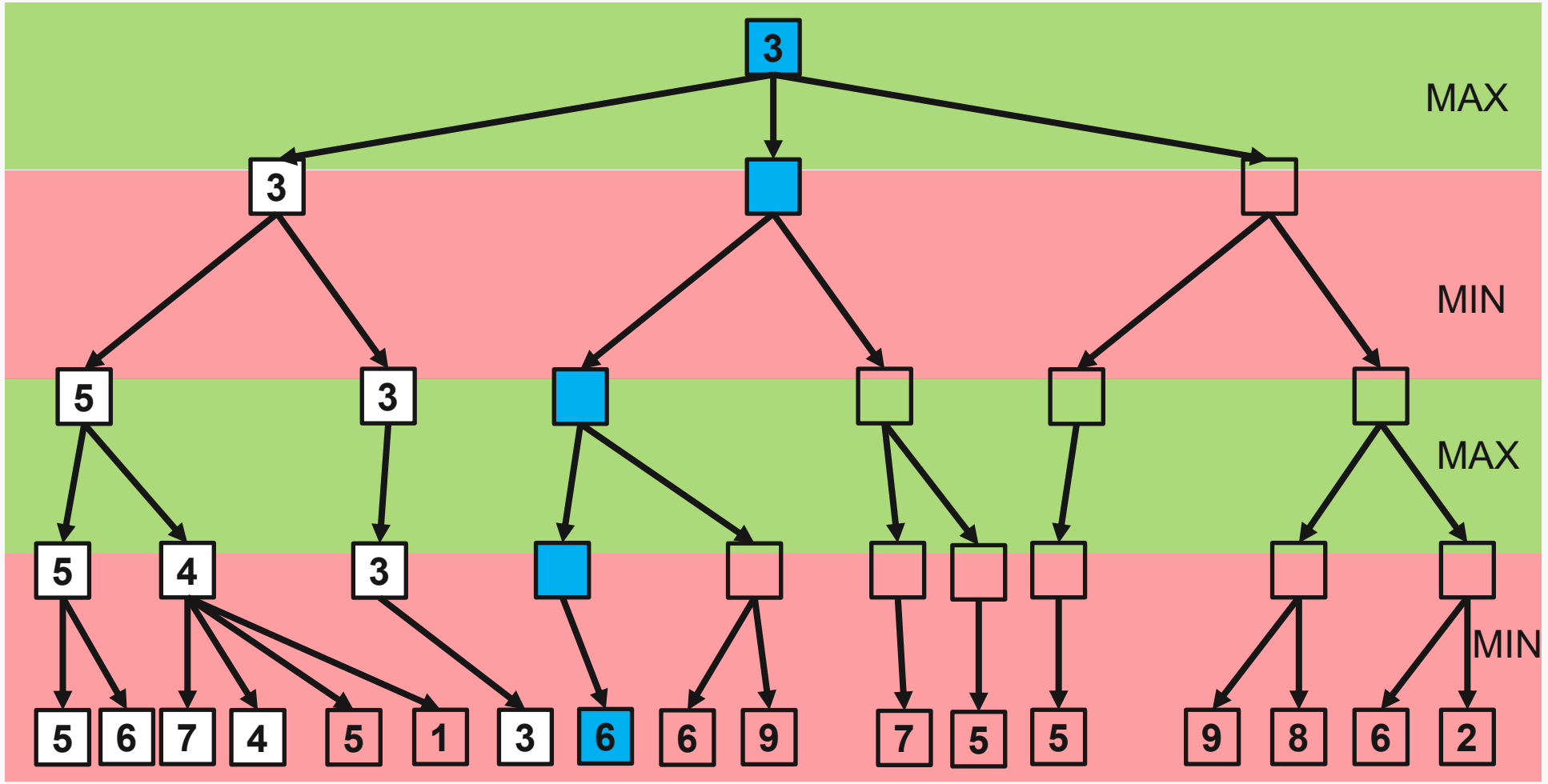
# AlphaBeta Prunning Ideea



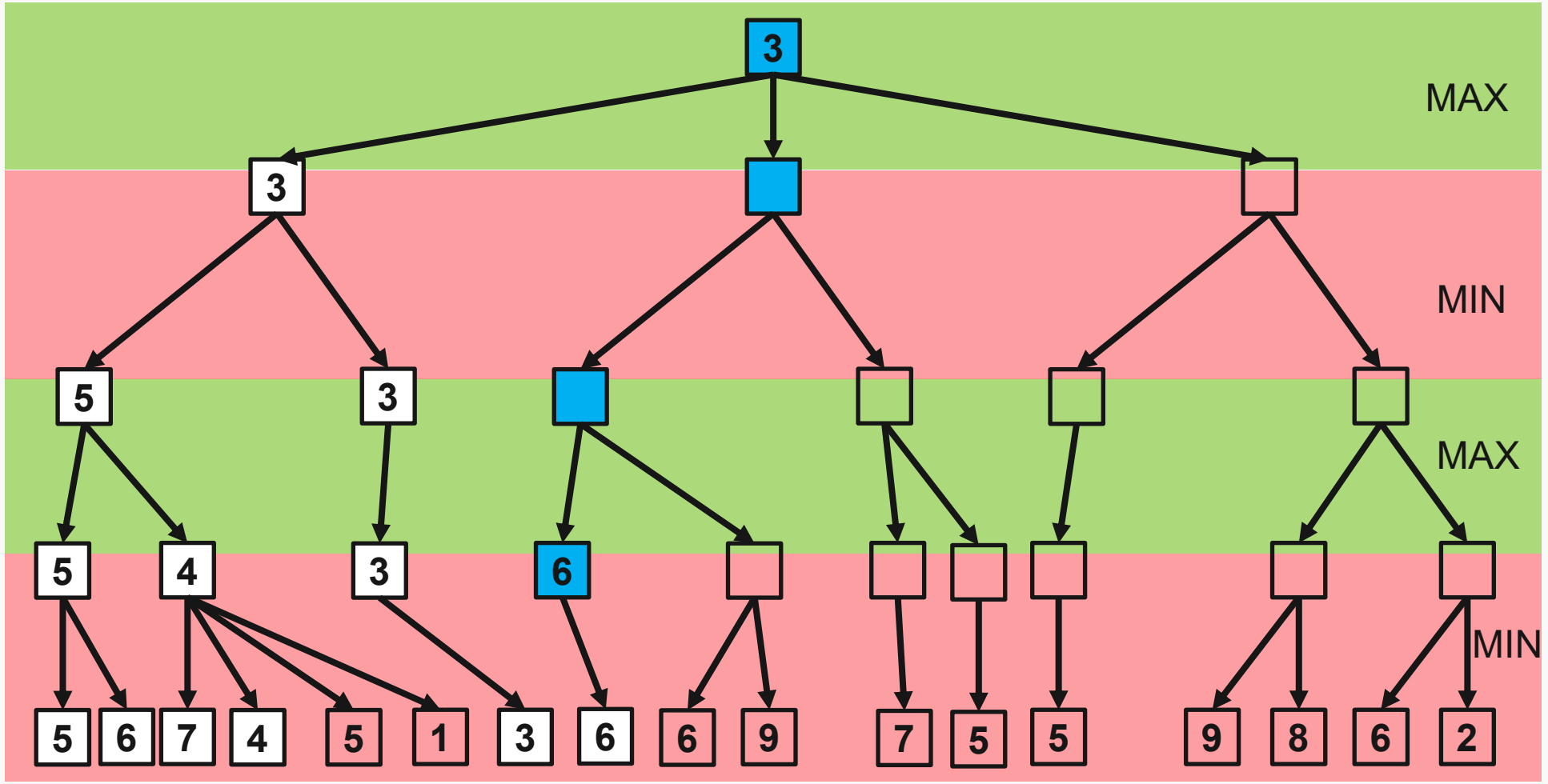
# AlphaBeta Prunning Ideea



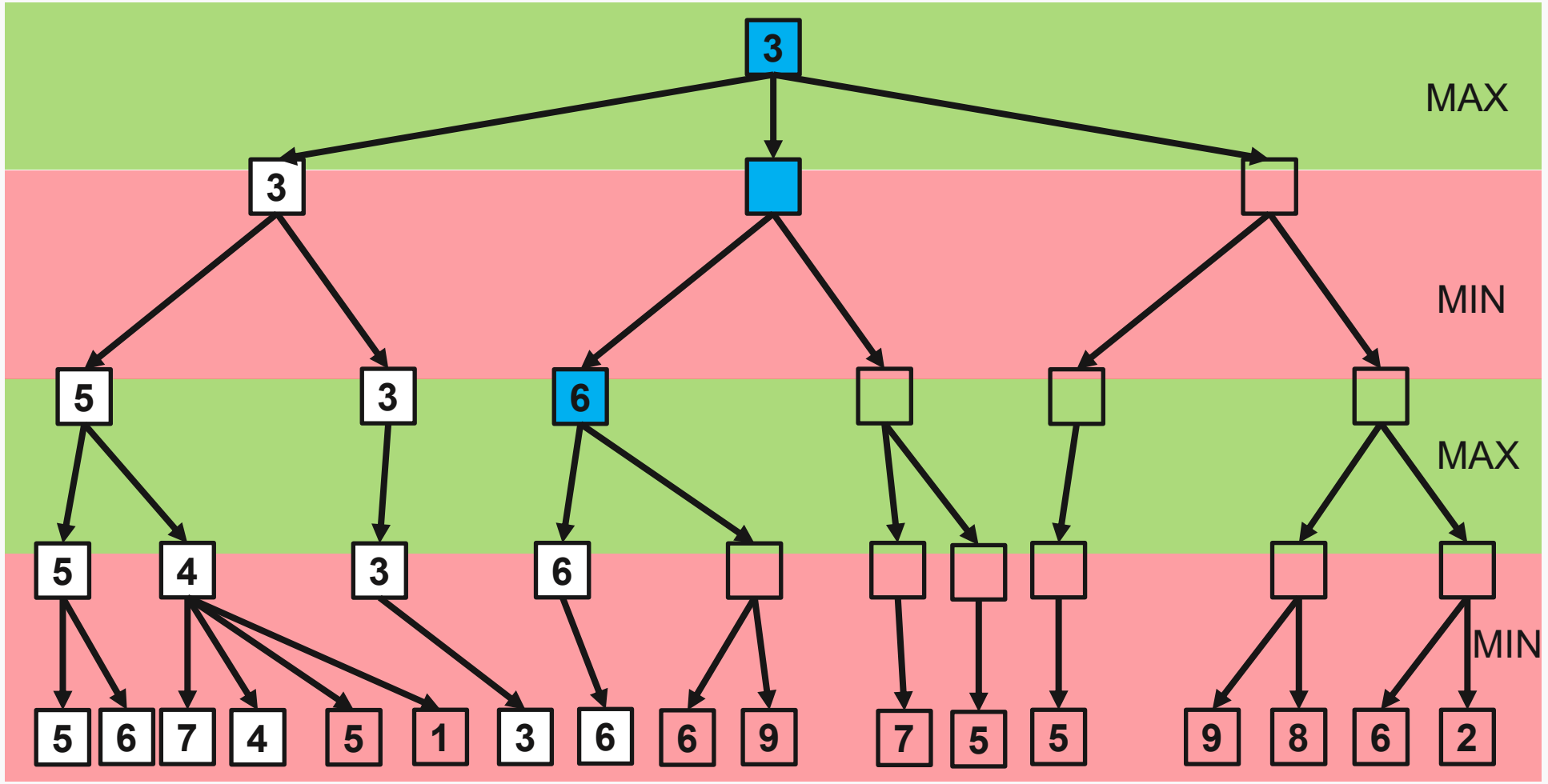
# AlphaBeta Prunning Ideea



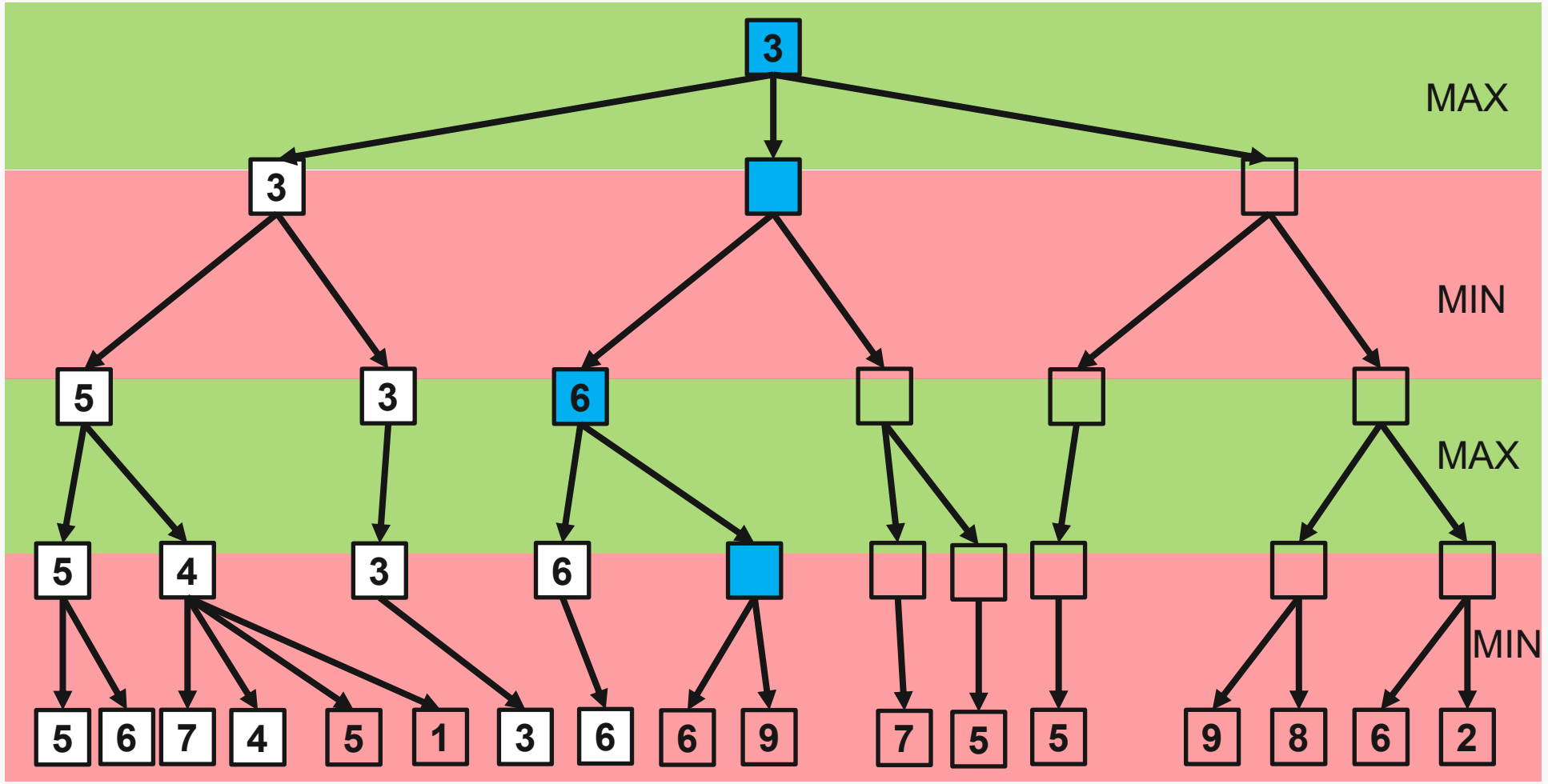
# AlphaBeta Prunning Ideea



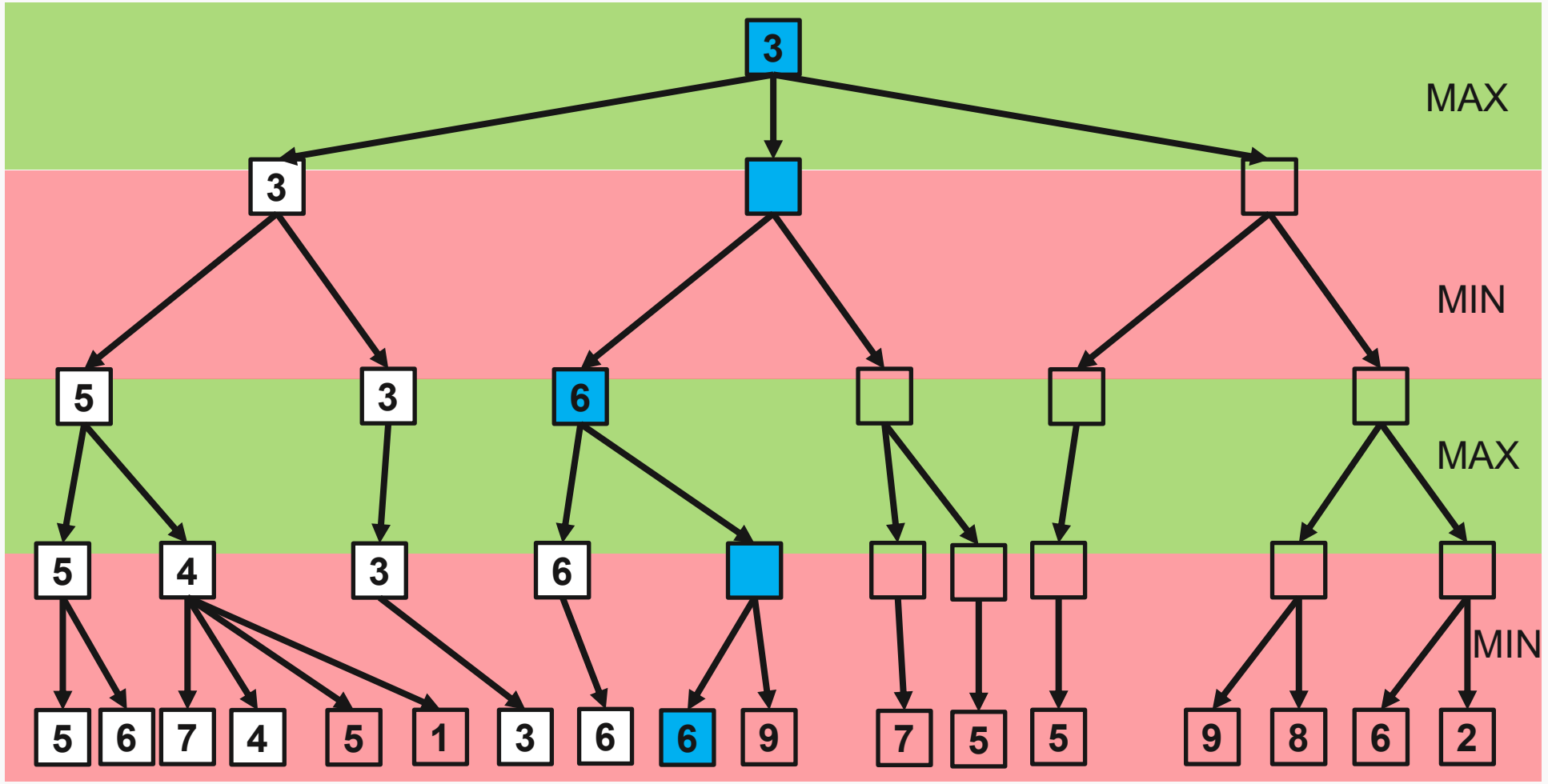
# AlphaBeta Prunning Ideea



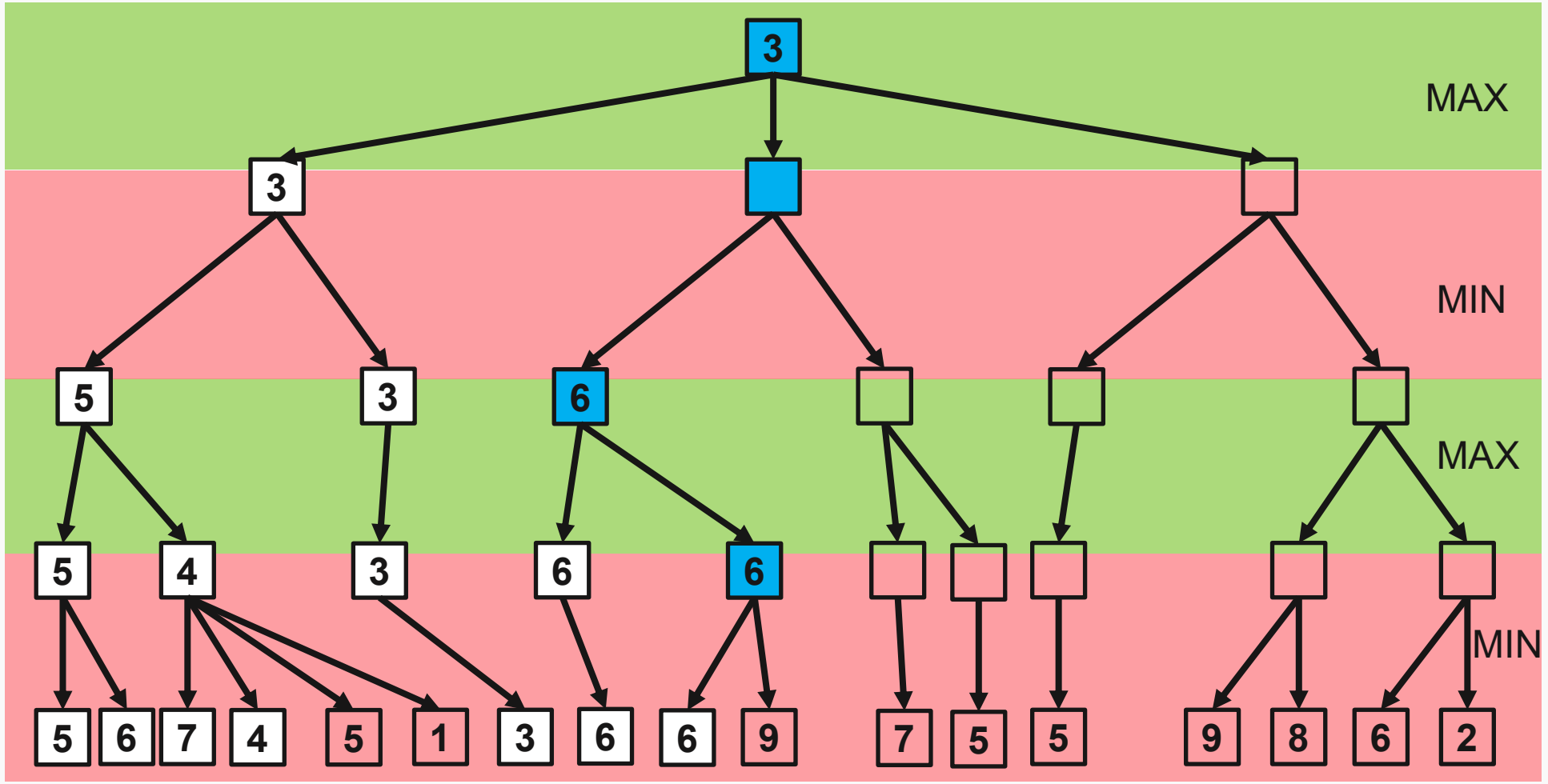
# AlphaBeta Prunning Ideea



# AlphaBeta Prunning Ideea

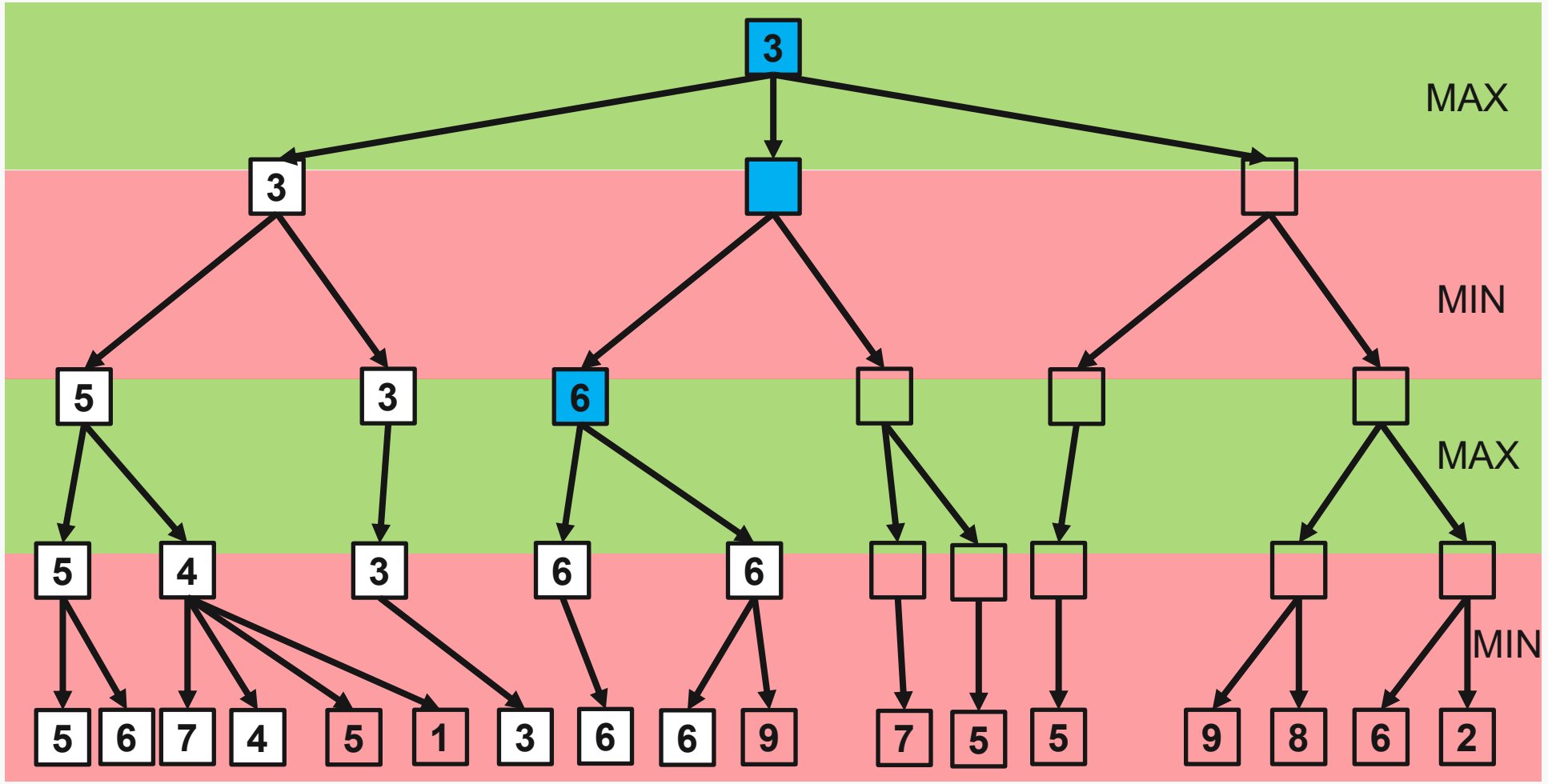


# AlphaBeta Prunning Ideea

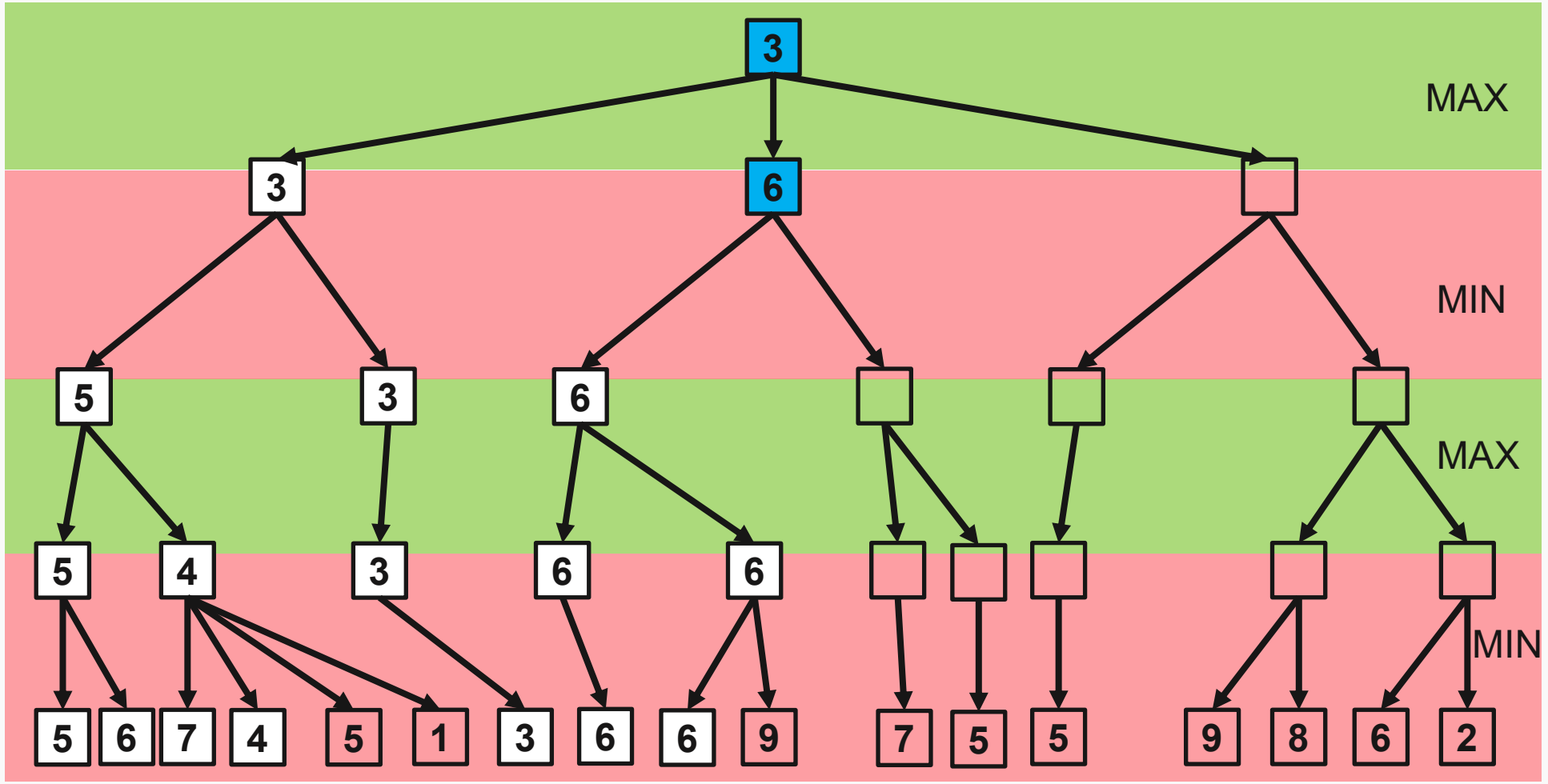



**Suntem deja siguri că la acest nivel competitorul va alege 6 sau mai mic.**
**Dar dacă el alege 6 sau mai mic, noi știm siguri că vom alege 6.**

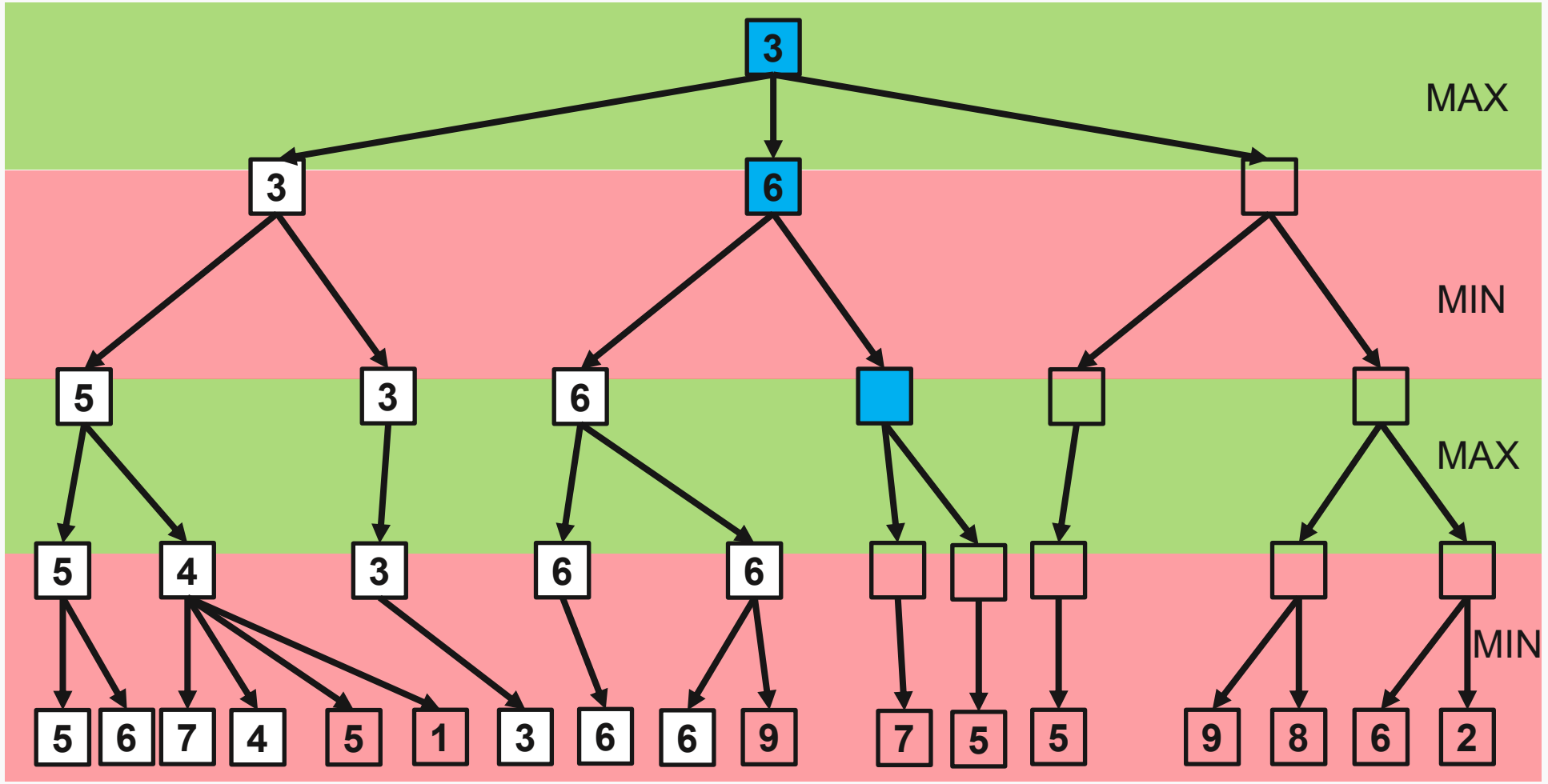
# AlphaBeta Prunning Ideea



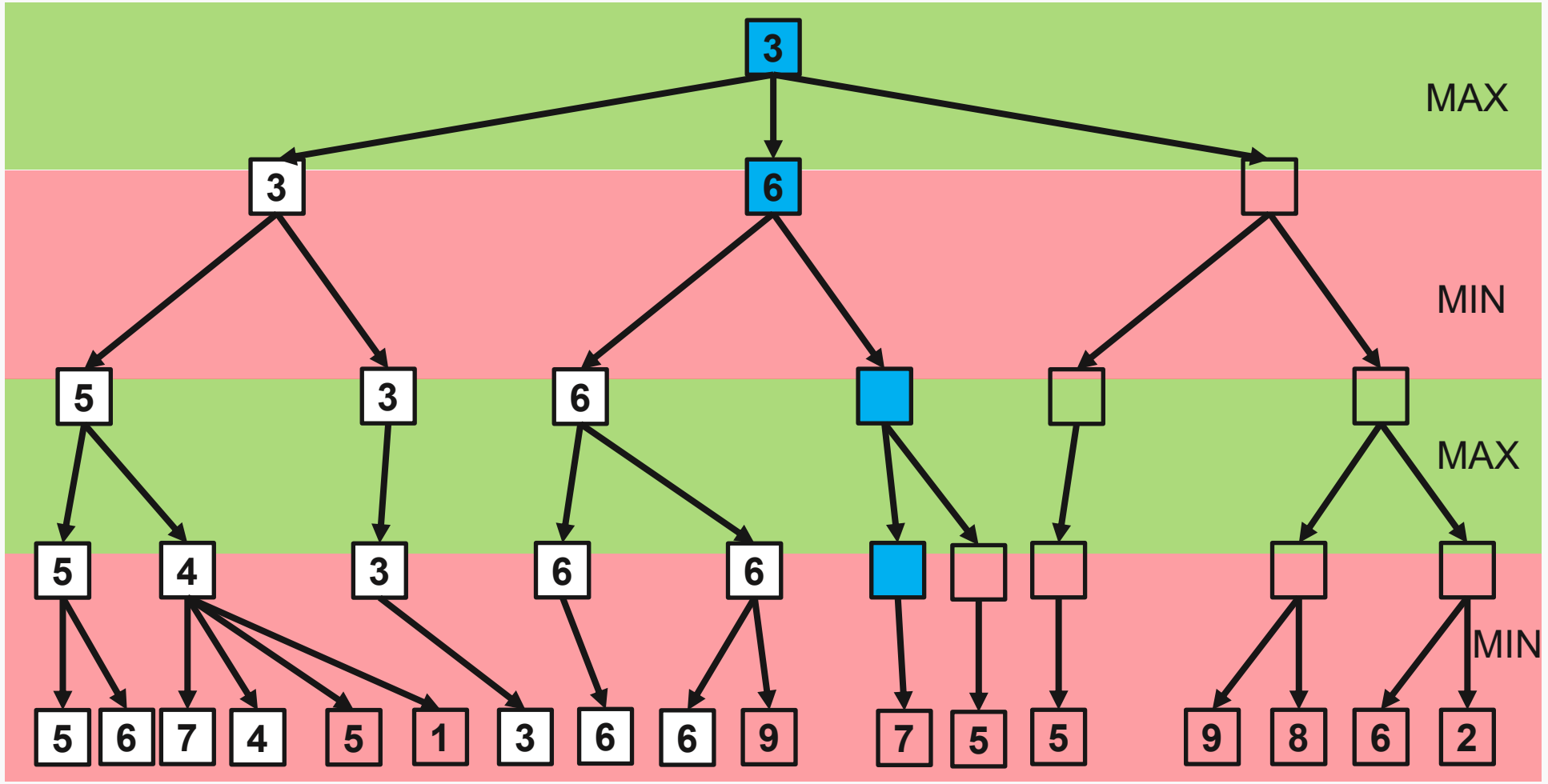
# AlphaBeta Prunning Ideea



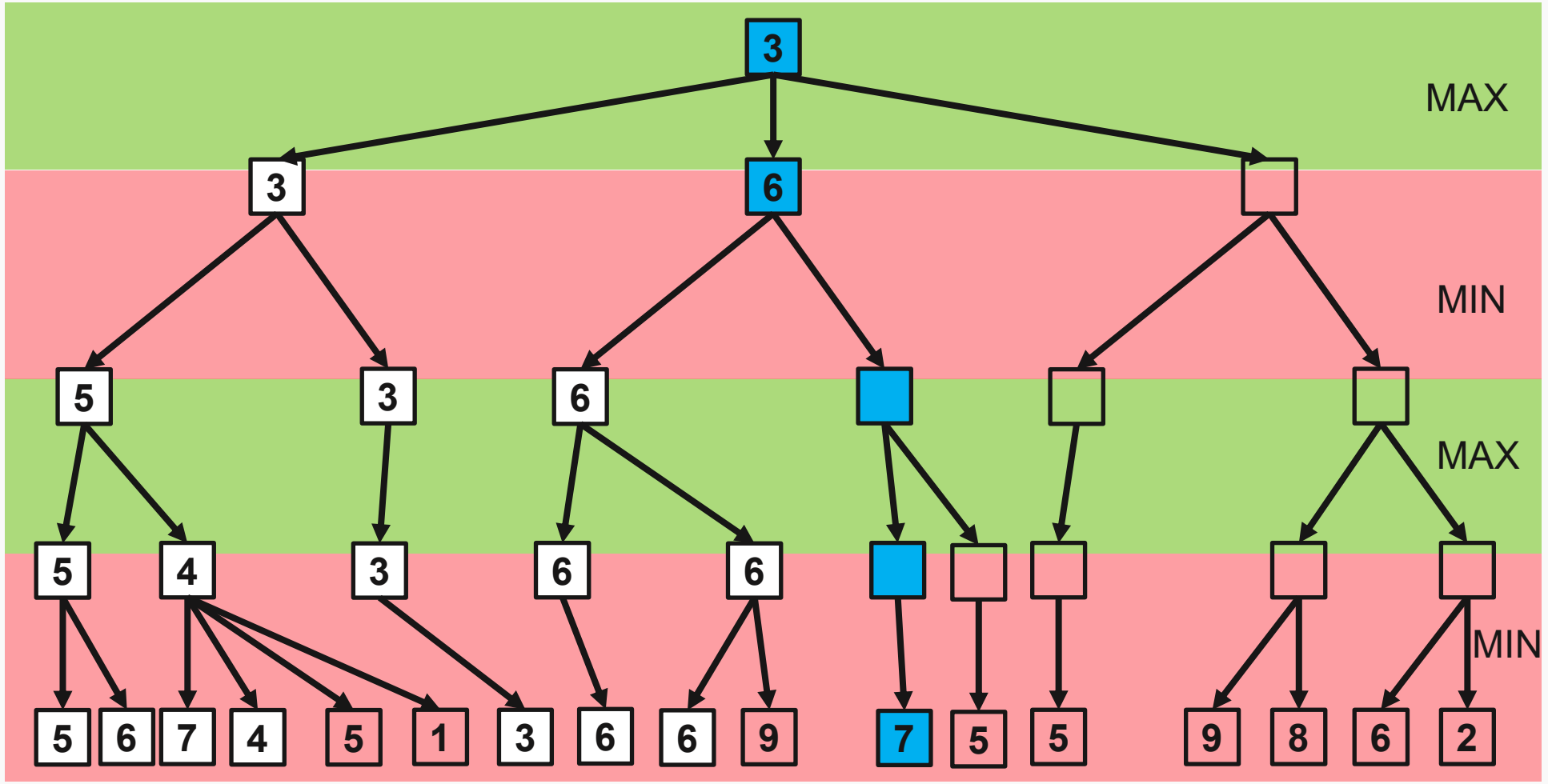
# AlphaBeta Prunning Ideea



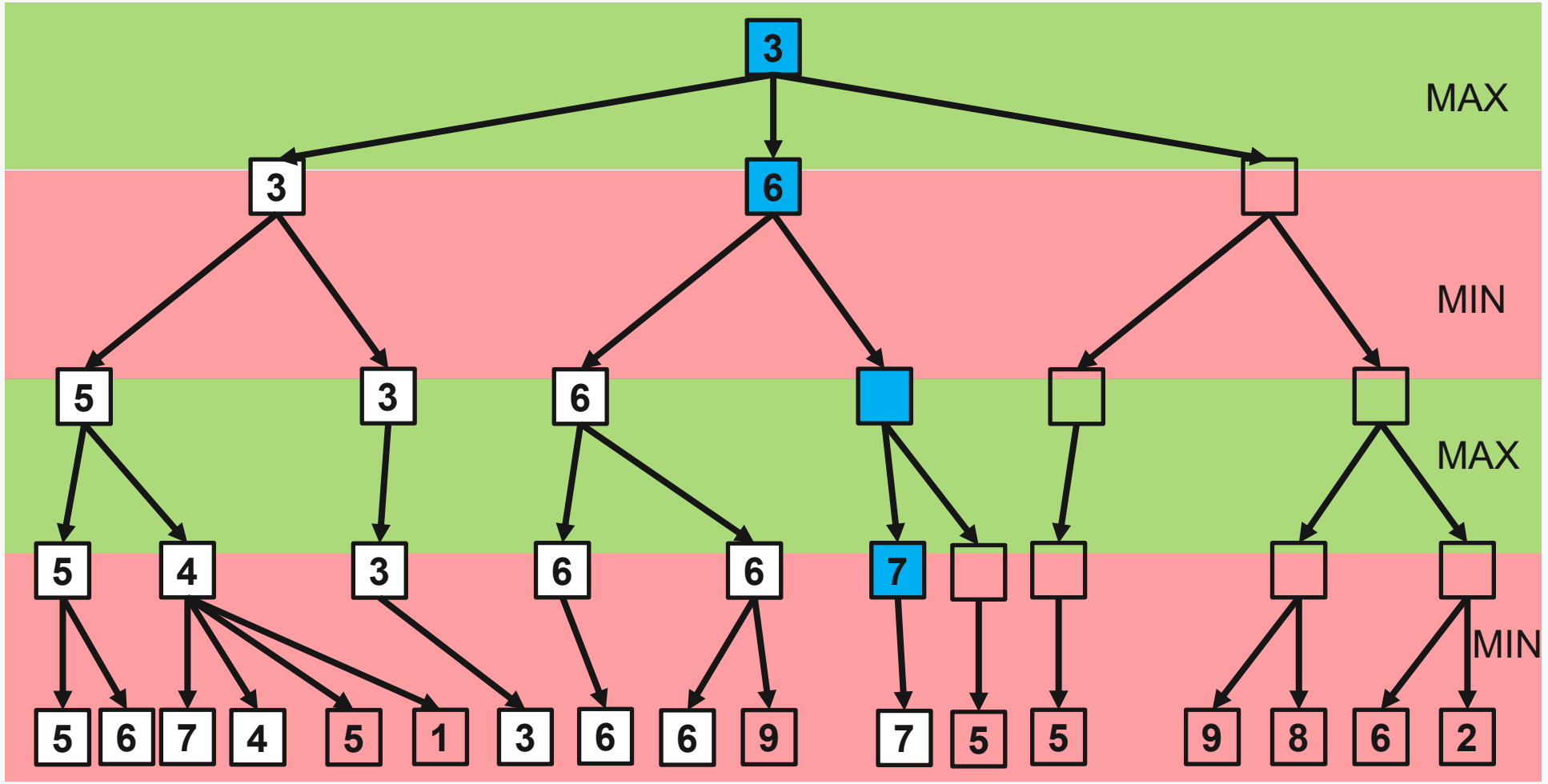
# AlphaBeta Prunning Ideea



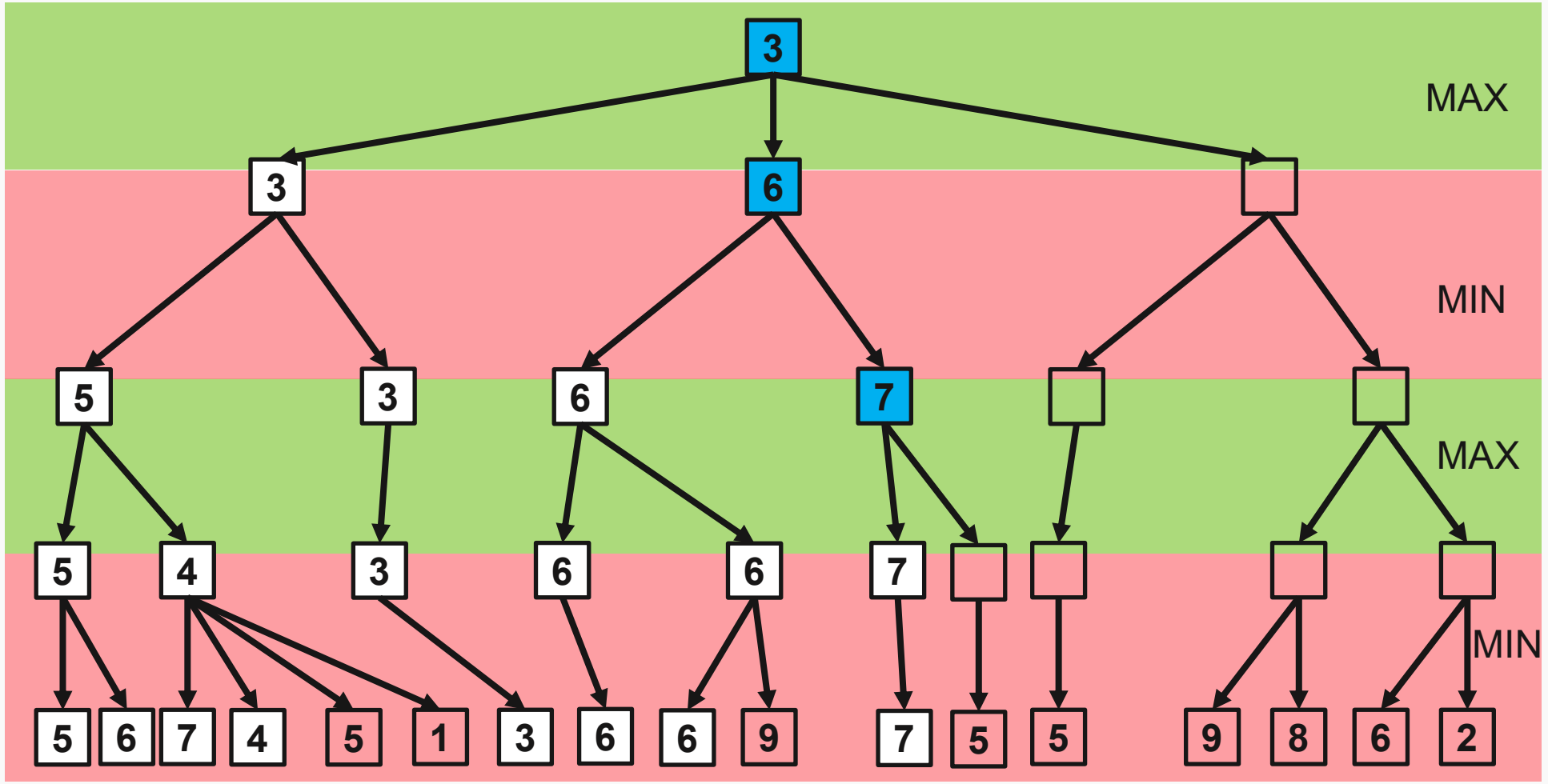
# AlphaBeta Prunning Ideea



# AlphaBeta Prunning Ideea

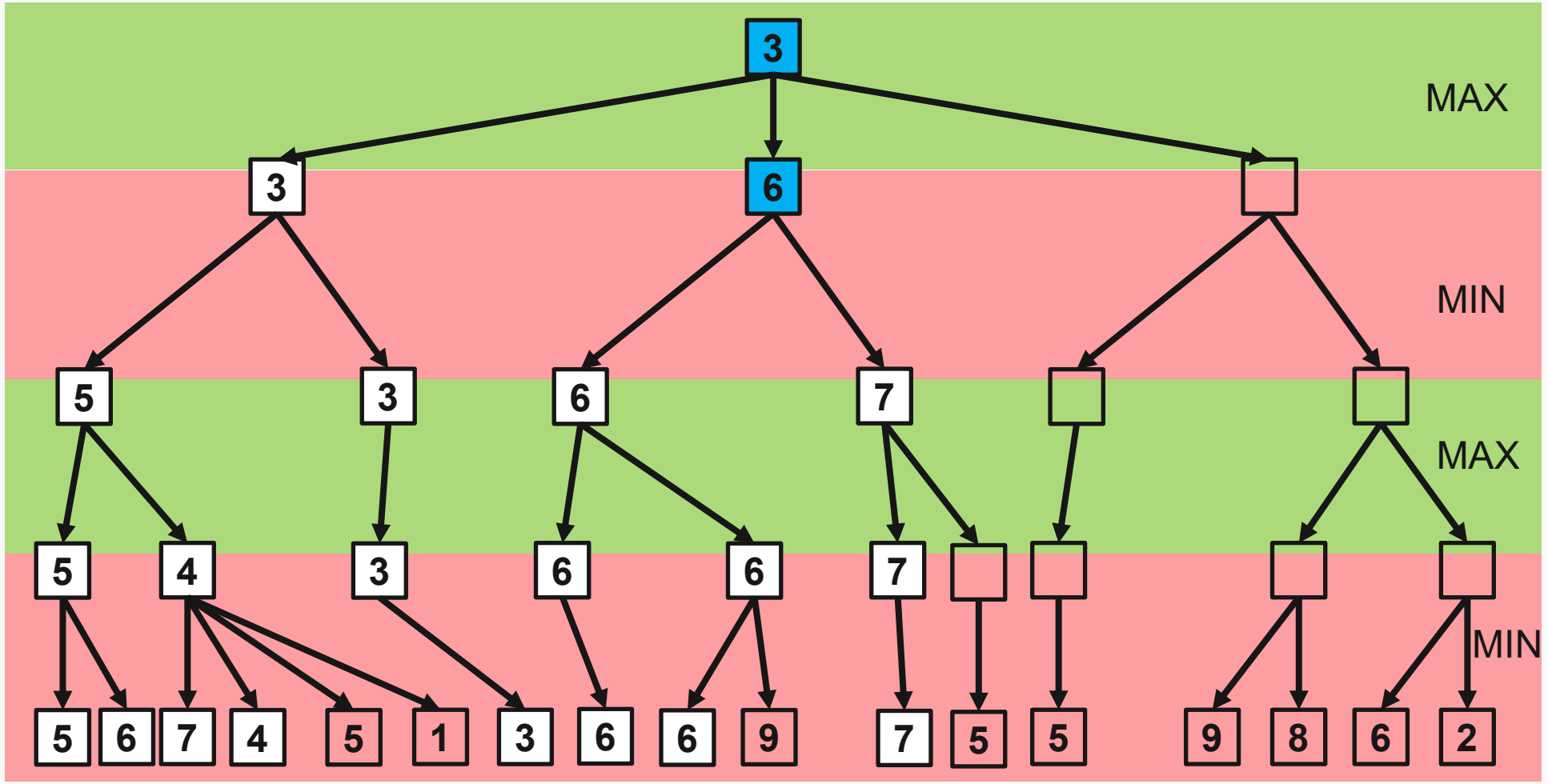


# AlphaBeta Prunning Ideea

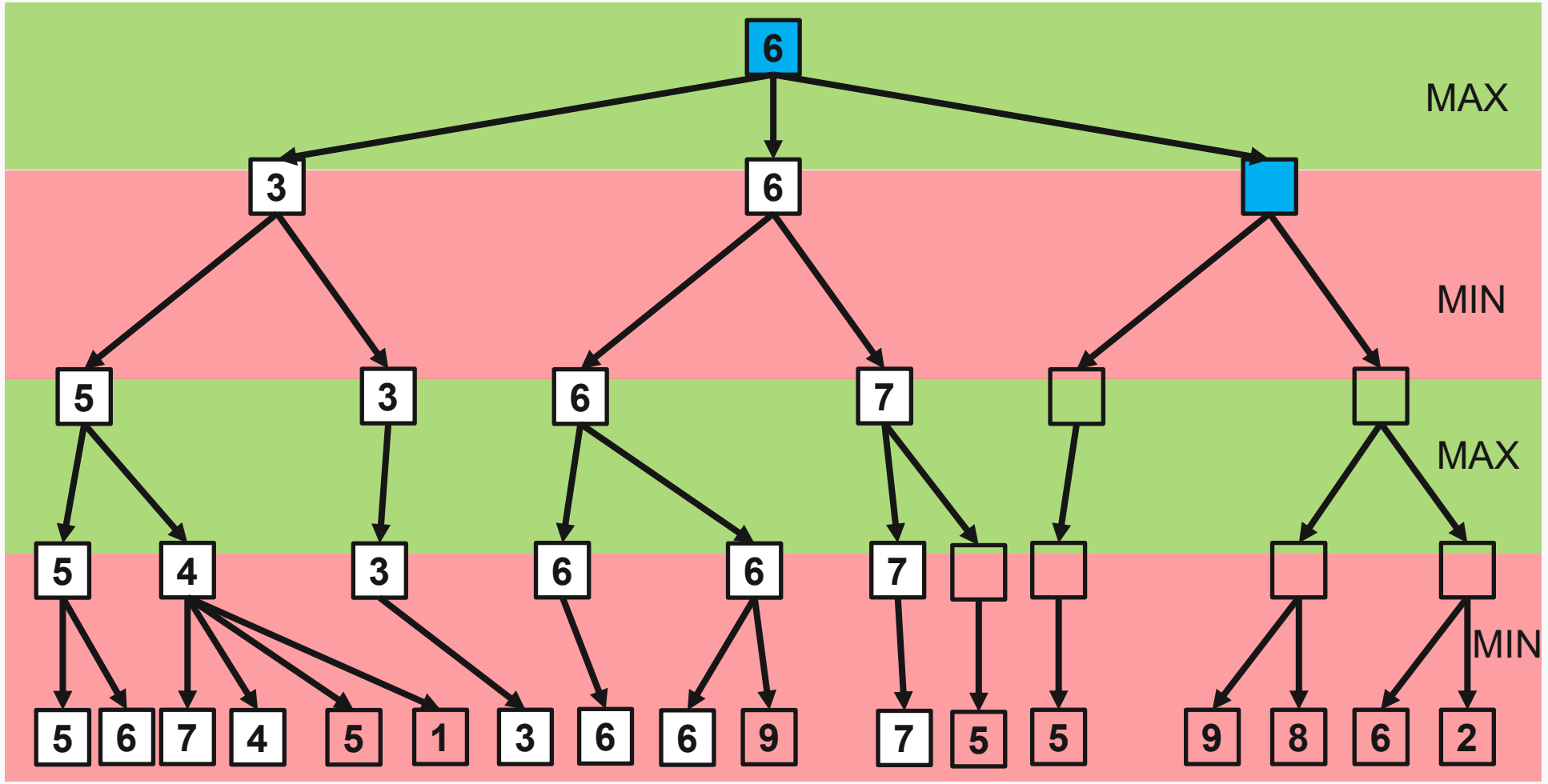



**Suntem deja siguri că la acest nivel noi vom alege 7 sau mai mare.**
**Dar dacă noi alegem 7 sau mai mare, știm că mai devreme competitorul alege 6.**

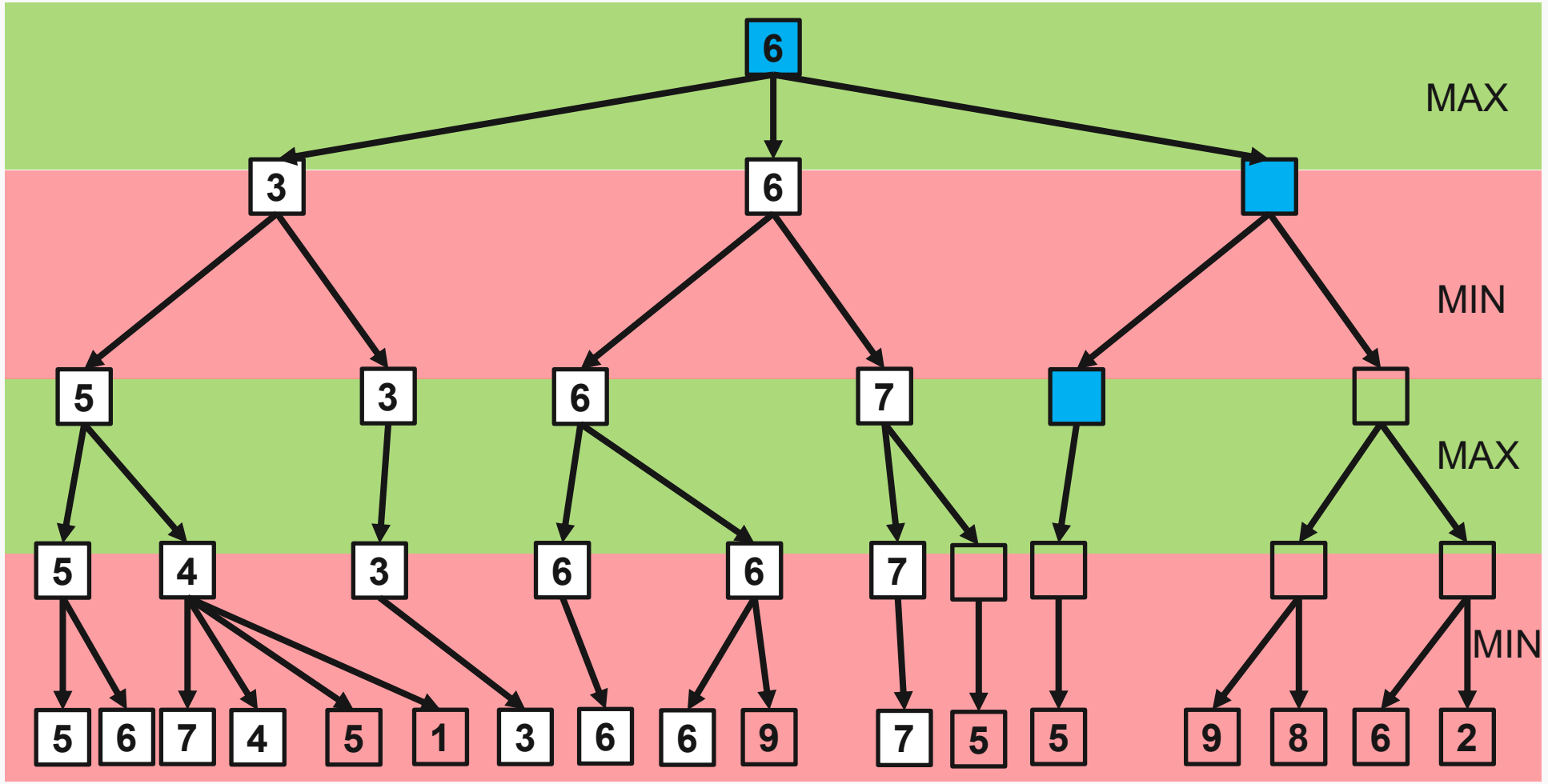
# AlphaBeta Prunning Ideea



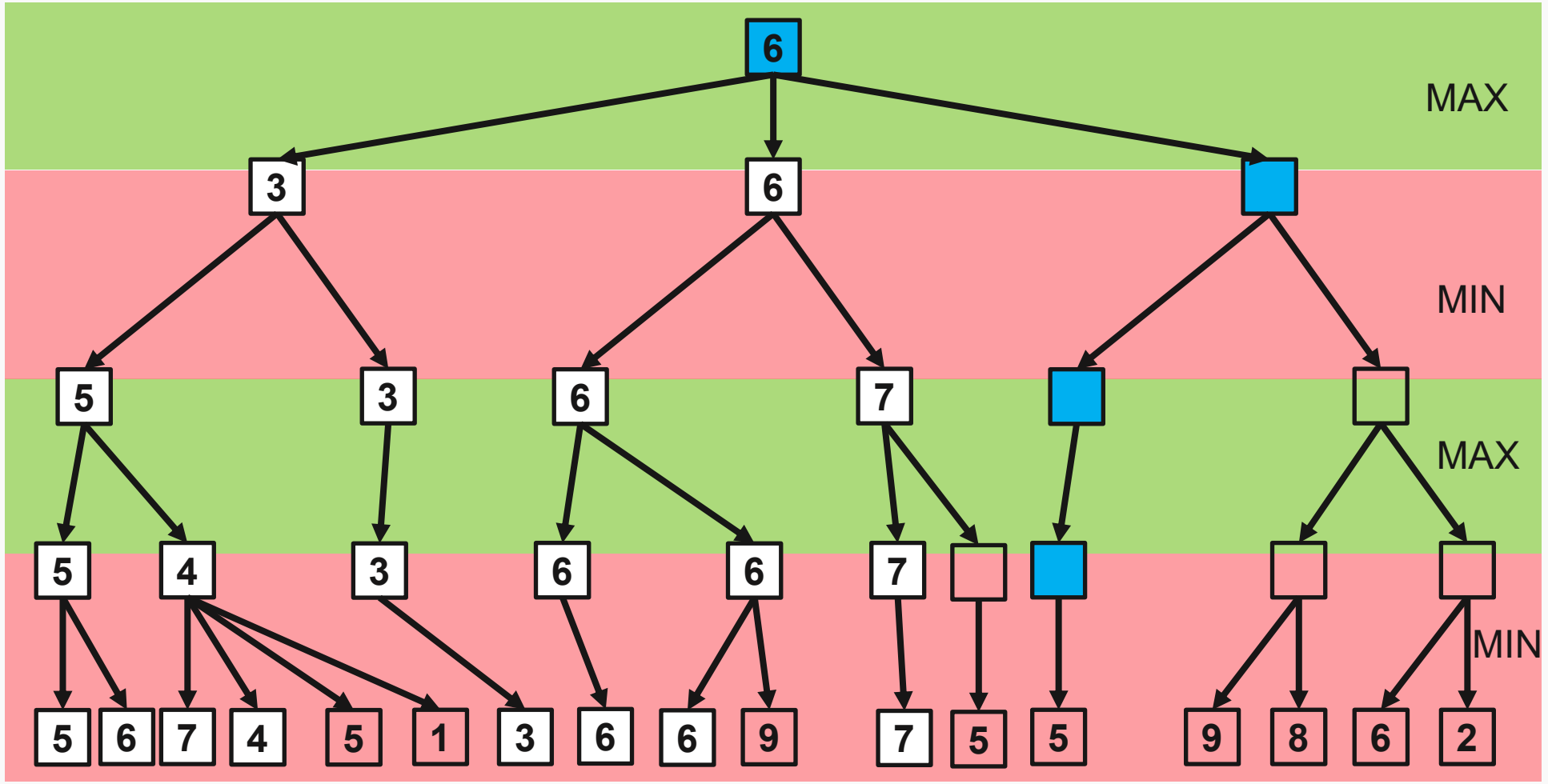
# AlphaBeta Prunning Ideea



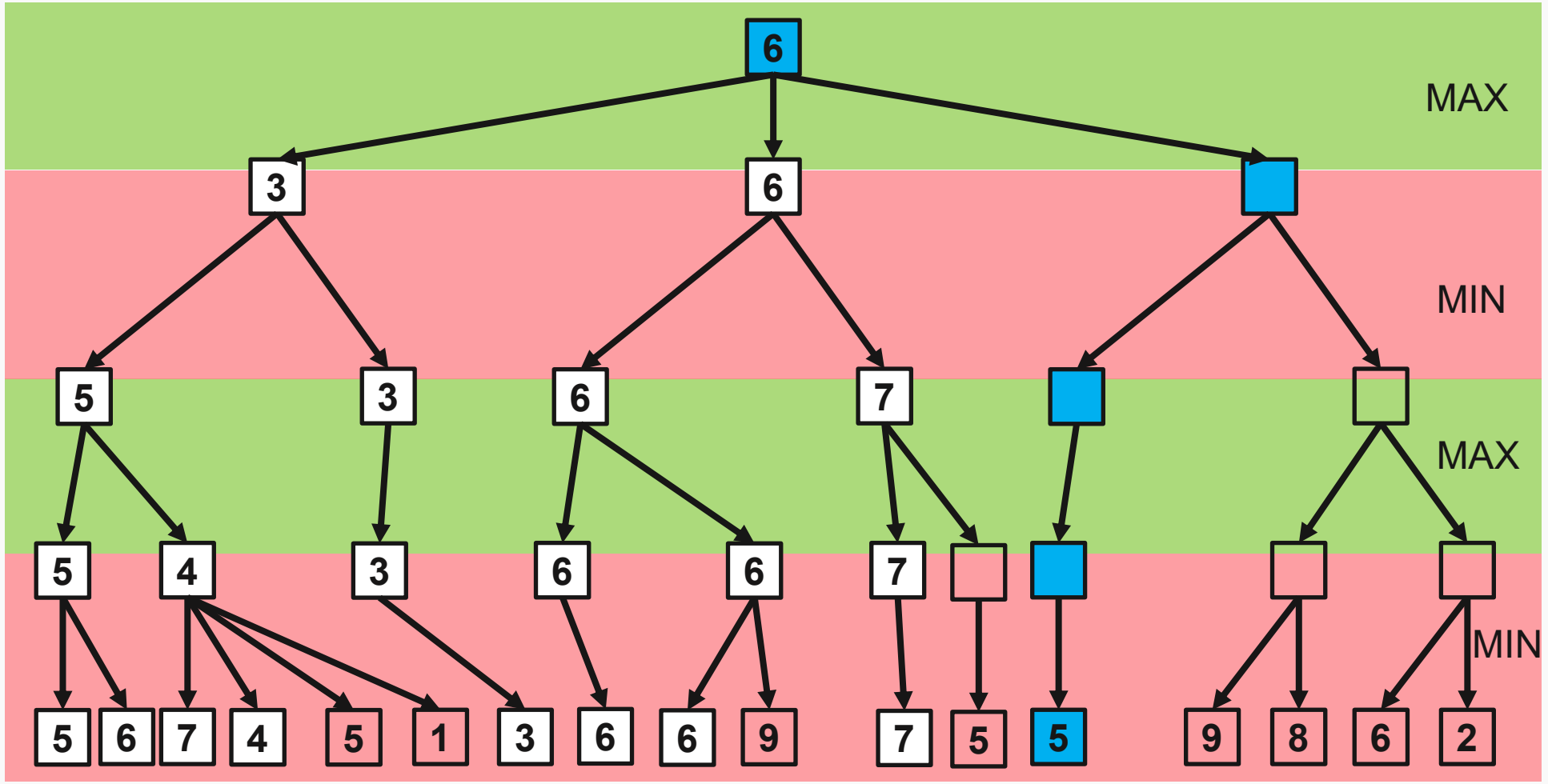
# AlphaBeta Prunning Ideea



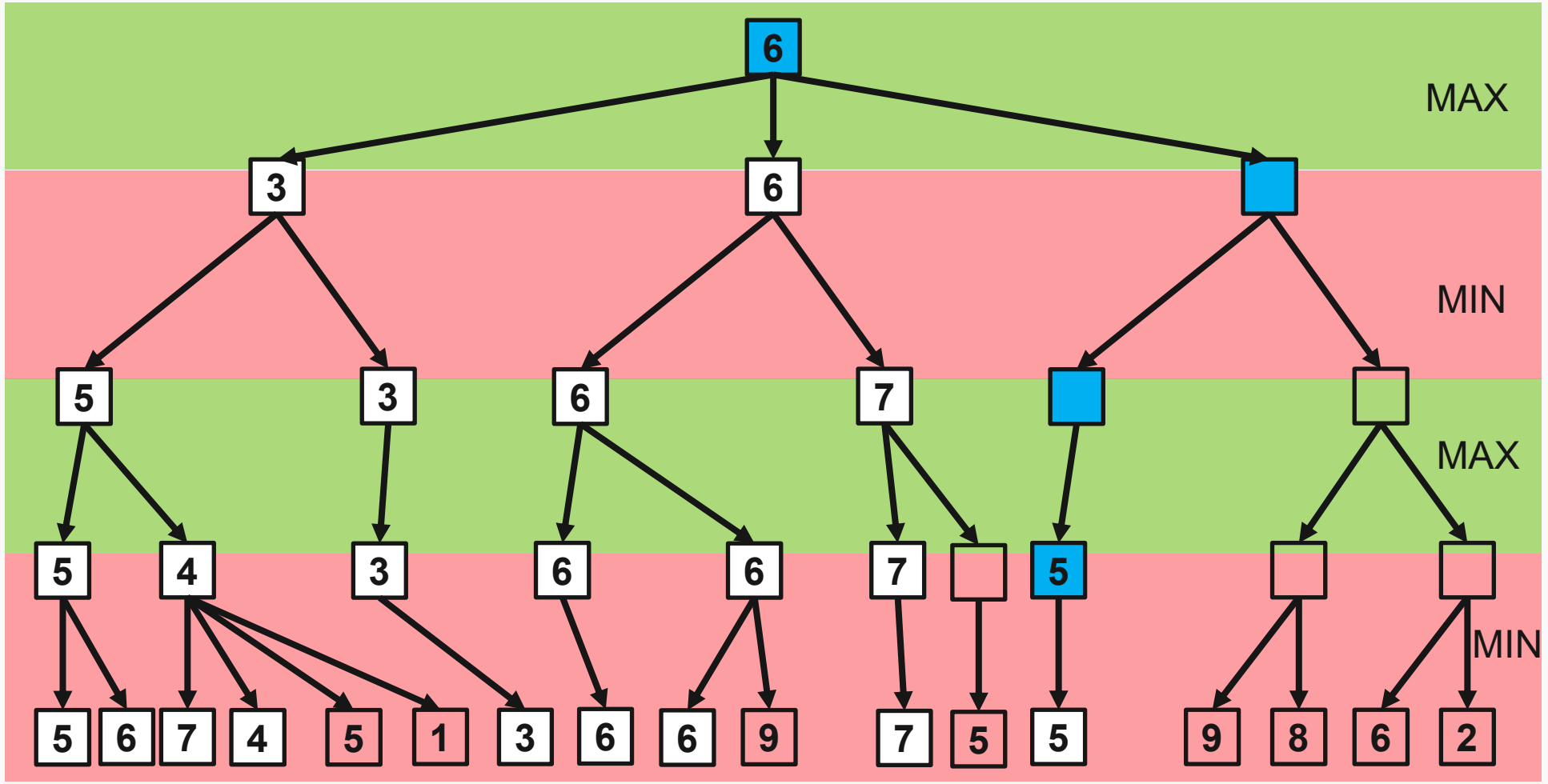
# AlphaBeta Prunning Ideea



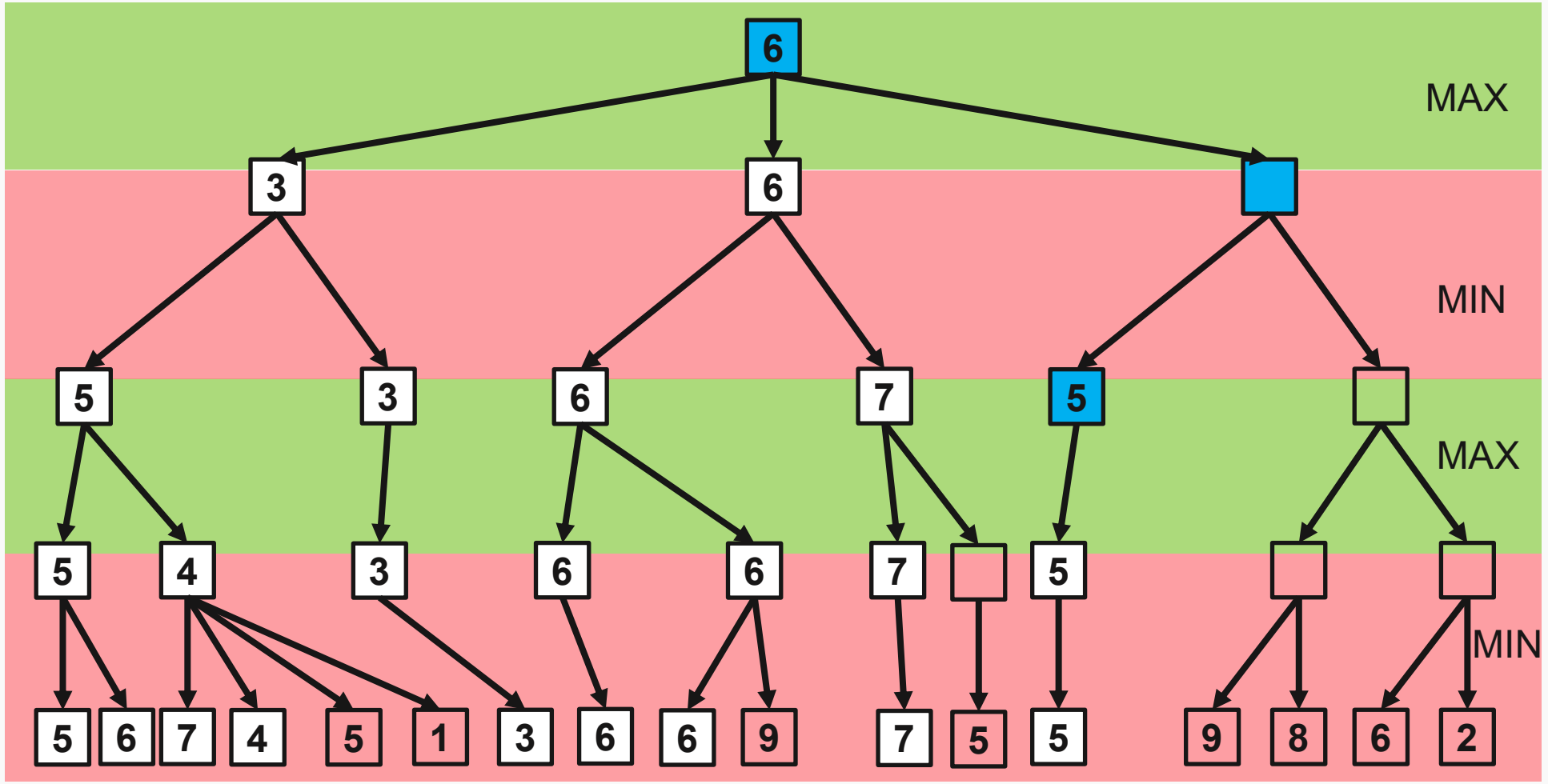
# AlphaBeta Prunning Ideea



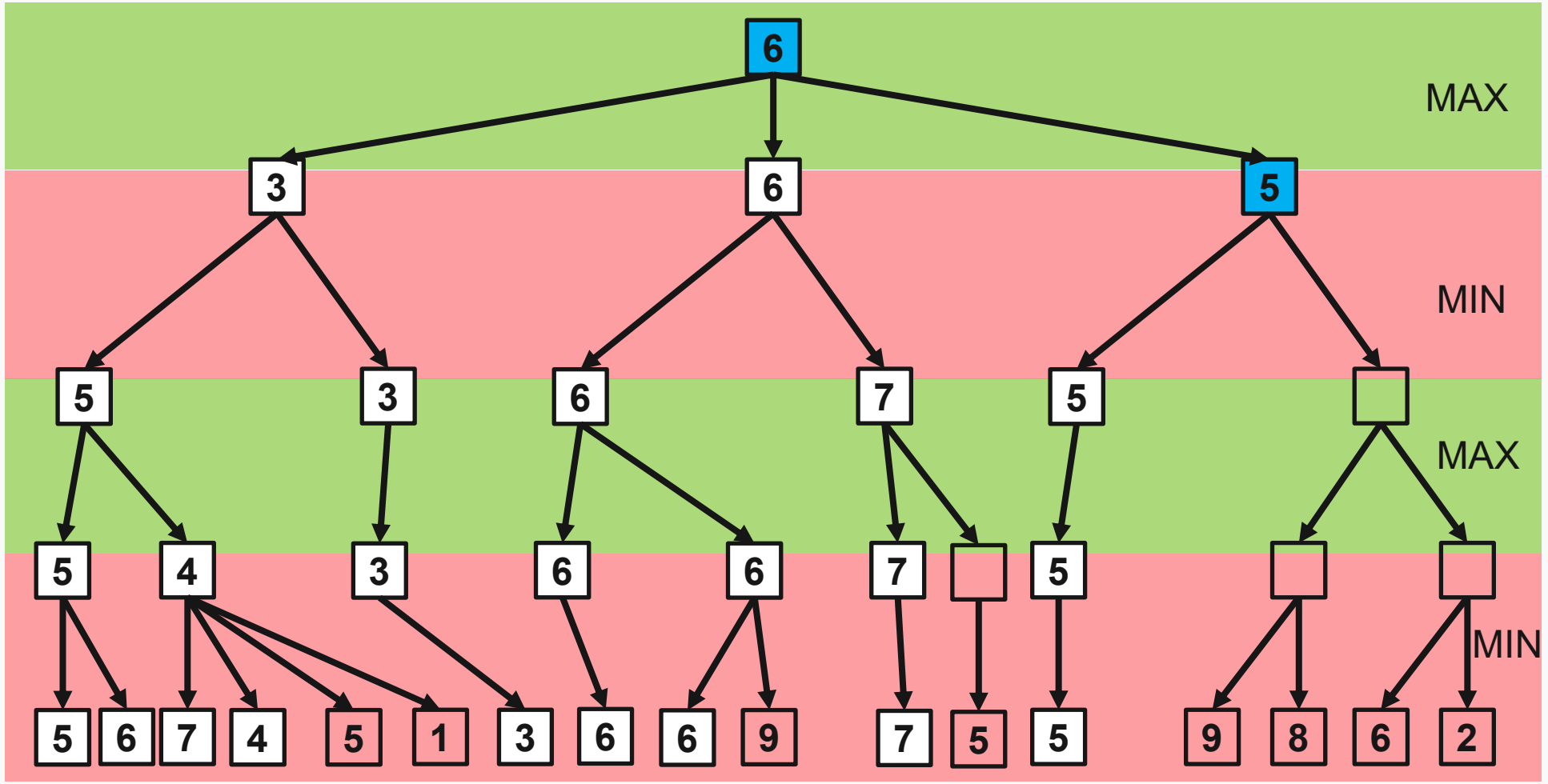
# AlphaBeta Prunning Ideea



# AlphaBeta Prunning Ideea

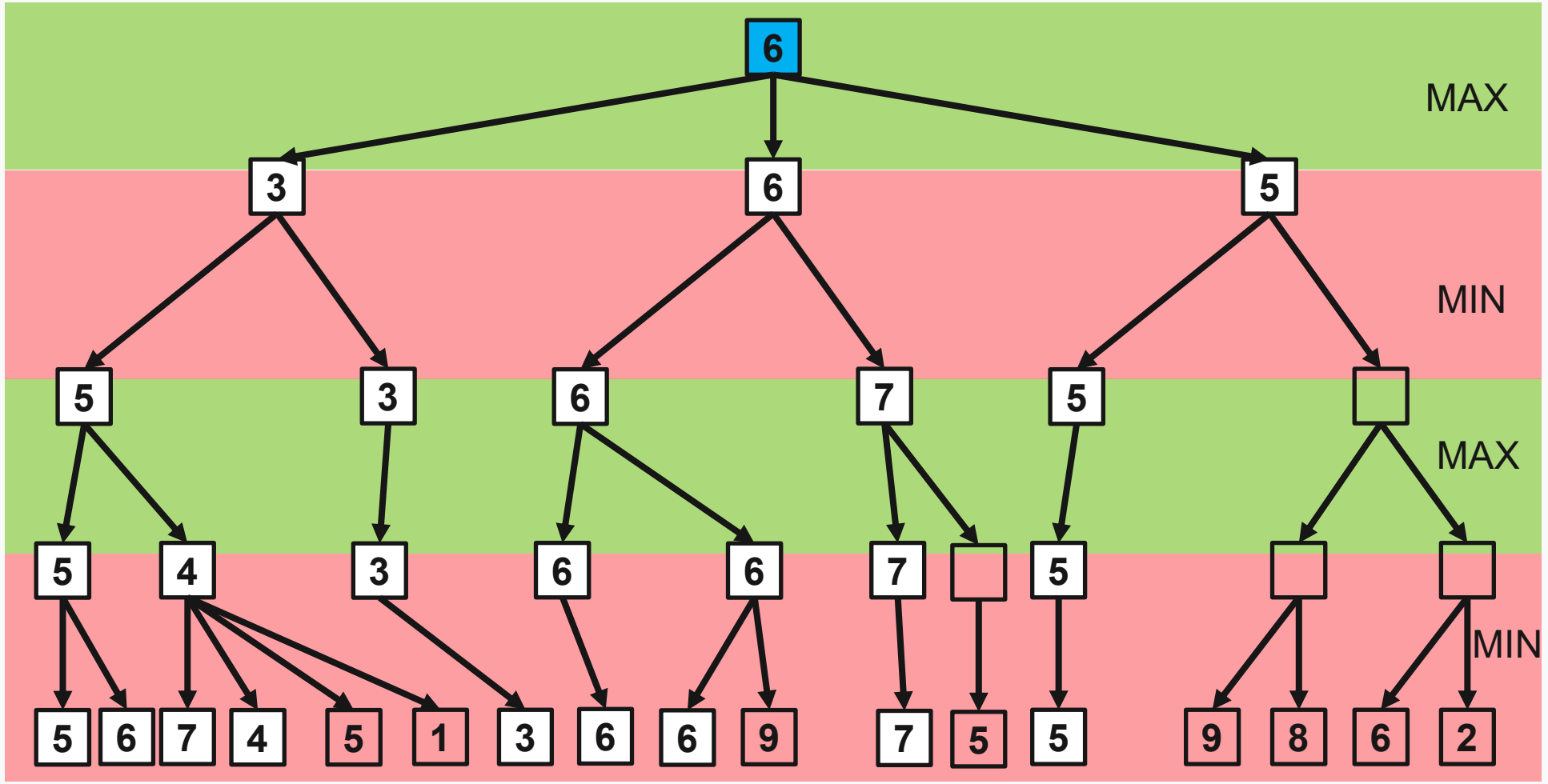


# AlphaBeta Prunning Ideea

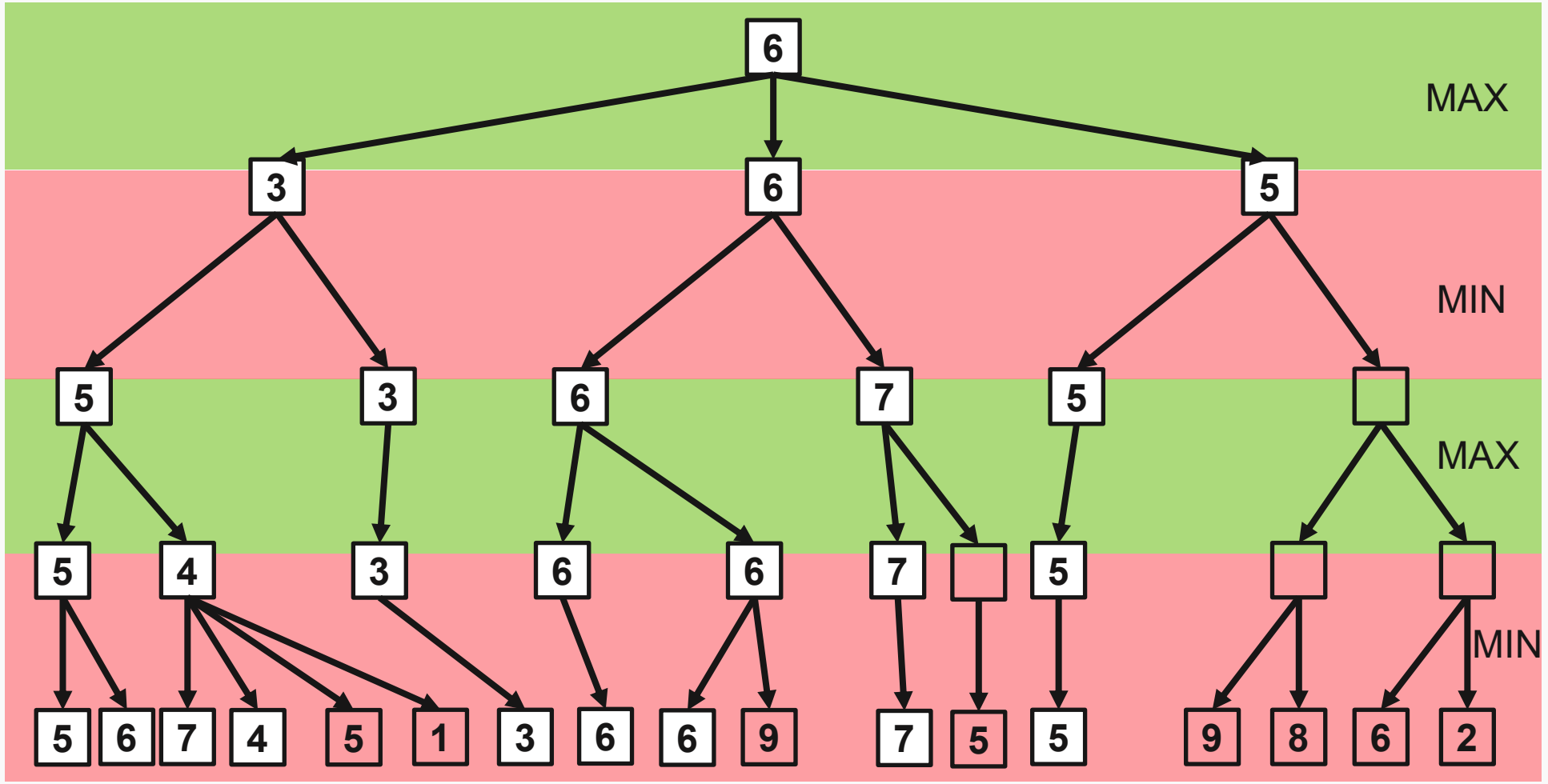



**Suntem deja siguri că la acest nivel competitorul va alege 5 sau mai mic.**
**Dar dacă el alege 5 sau mai mic, noi știm siguri că vom alege 6.**

# AlphaBeta Prunning Ideea



# AlphaBeta Prunning Ideea



# MinMax + AlphaBeta Pruning

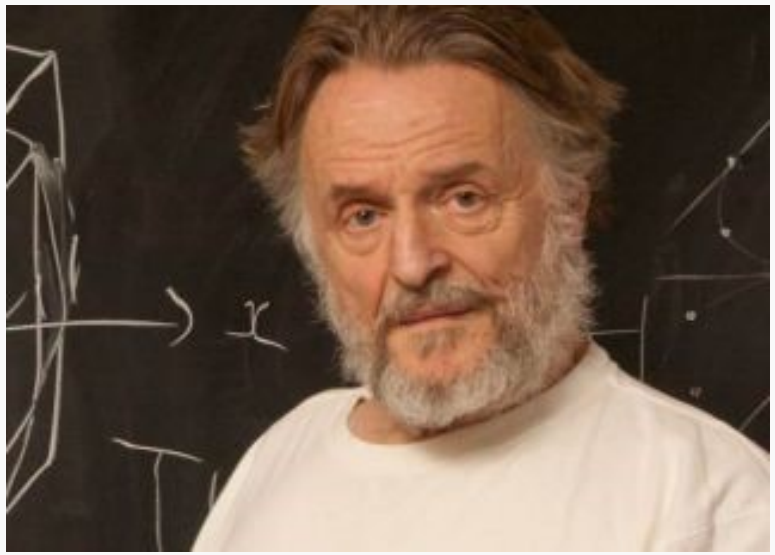
```
int alphaBeta(gameNode node, int level, int alpha, int beta) {
  if (level == maxDepth ||  isEndGame(node))
        return heuristic(node);
  if (level % 2 == maximizingPlayer) {
    int value = −∞;
    for each (child of node) {
      value = max(value, alphaBeta(child, level + 1, alpha, beta));
      alpha = max(alpha, value);
      if (alpha >= beta)
        break;
    }
    return value;
  } else {
    int value = +∞;
    for each (child of node) {
      value = min(value, alphaBeta(child, level + 1, alpha, beta));
      beta = min(beta, value);
      if (alpha >= beta)
        break;
    }
    return value;
  }
}
```

# Apelare AlphaBeta

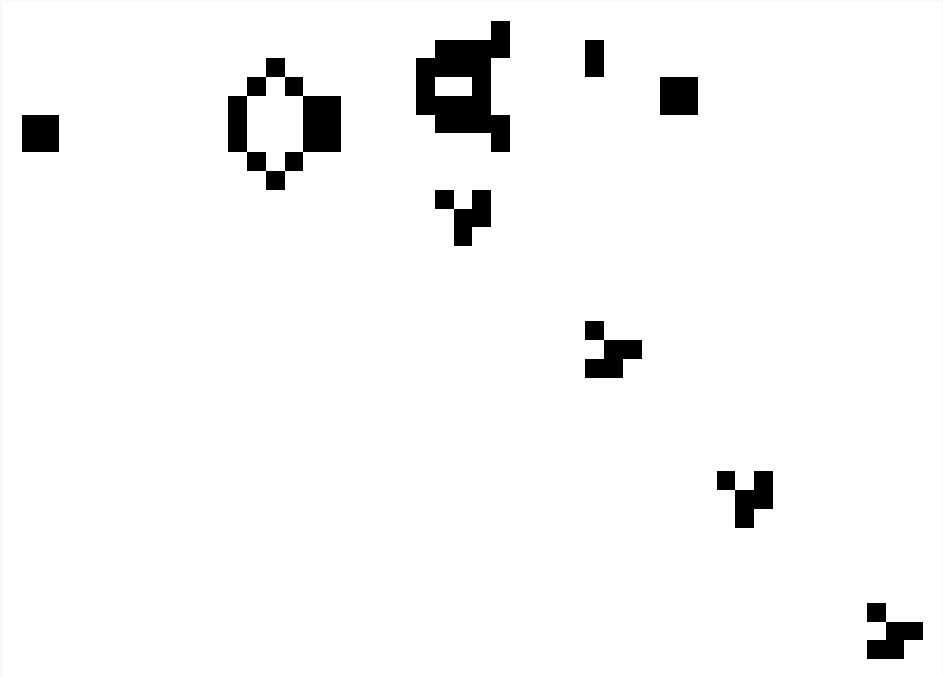
```
for each (child of node)
     score[child] = alphaBeta(child, 0, -∞, +∞));
```

Mutarea este acel child pentru care score e maxim

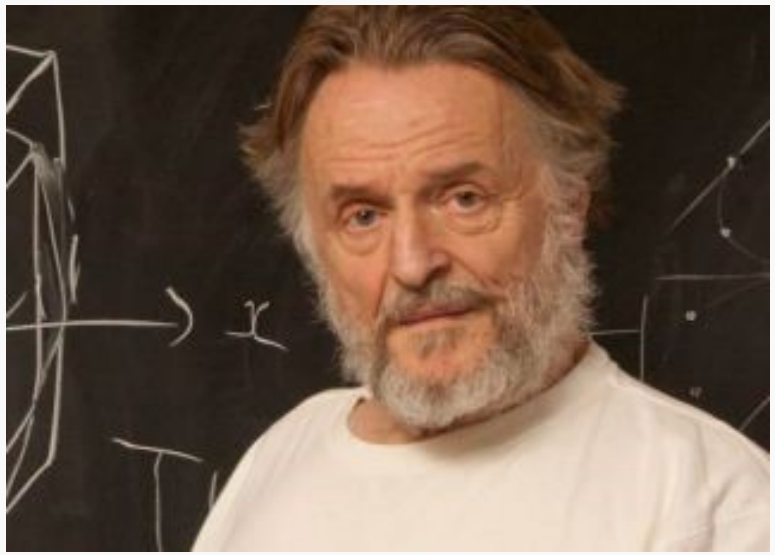
# John Conway – game of life



26-Decembrie-1937 -> 11-Aprilie-2020



# John Conway – game of life



Celulele unei matrici pot avea valori **0** sau **1**
Orice celulă **1** fără vecini devine **0**
Orice celulă **1** cu 1 vecin devine **0**
Orice celulă **1** cu 2 vecini rămâne **1**
Orice celulă **1** cu 3 vecini rămâne **1**
Orice celulă **1** cu 4+ vevini devine **0**
Orice celulă **0** cu 3 vecini devine **1**

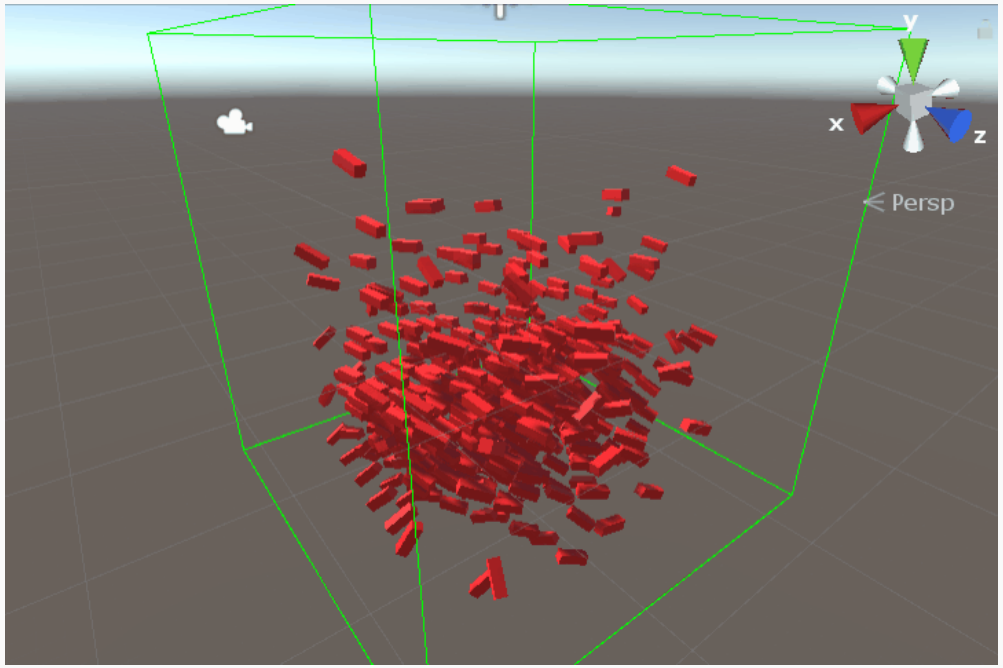
26-Decembrie-1937 -> 11-Aprilie-2020
Cause of death: Coronavirus

https://bitstorm.org/gameoflife/

# Pentru curioși – John Conway – surreal numbers

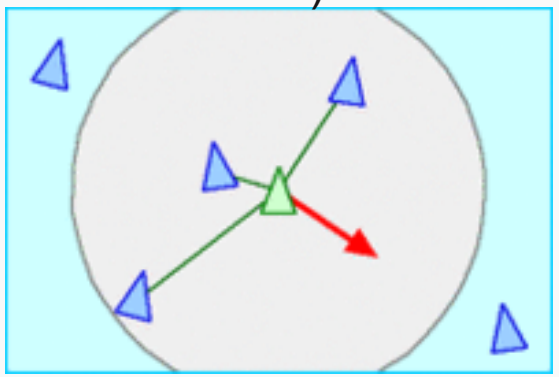
- https://en.wikipedia.org/wiki/Surreal_number
- https://www.youtube.com/watch?v=E8kUJL04ELA
- https://www.youtube.com/watch?v=1eAmxgINXrE
- https://www.youtube.com/watch?v=R9Plq-D1gEk

# Boids - Craig Reynolds

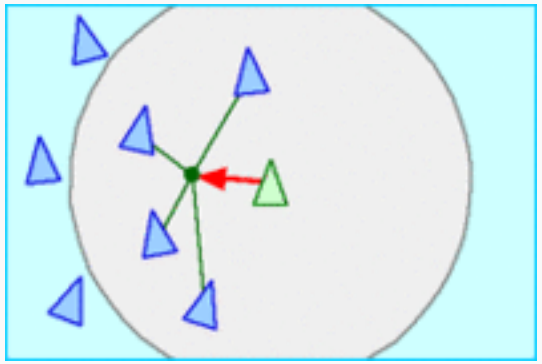


# Boids – Reguli simple comportament complex

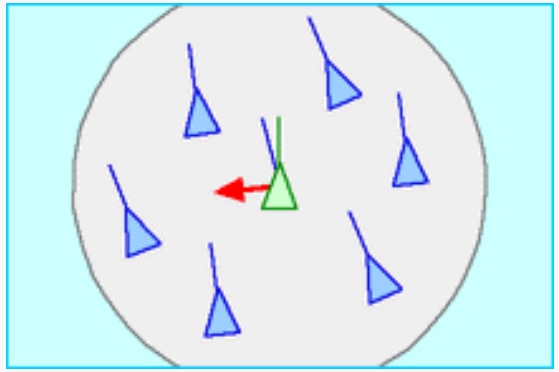
Separare (incearca sa evite coliziuni)



Coeziune (sta aproape de centrul grupului)



Aliniere (zboara in aceeasi directie cu directia generala a grupului)



# Further discussion

- Nash equilibrium + Pareto Optimal
  - Prisoner's Dilema
  - Movie: A beautiful mind

- Monte Carlo + Las Vegas (random algorithms)