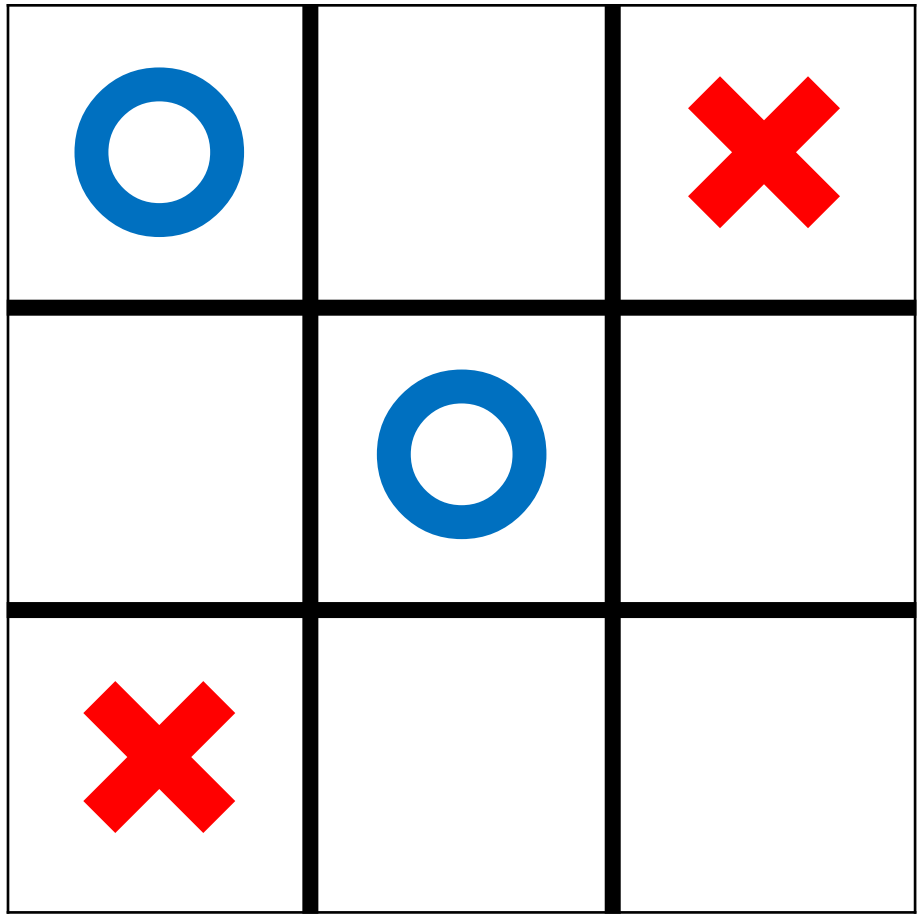
KI-Algorithmen für 2-Personen-Spiele: Minimax

X ist am Zug. Wer gewinnt?



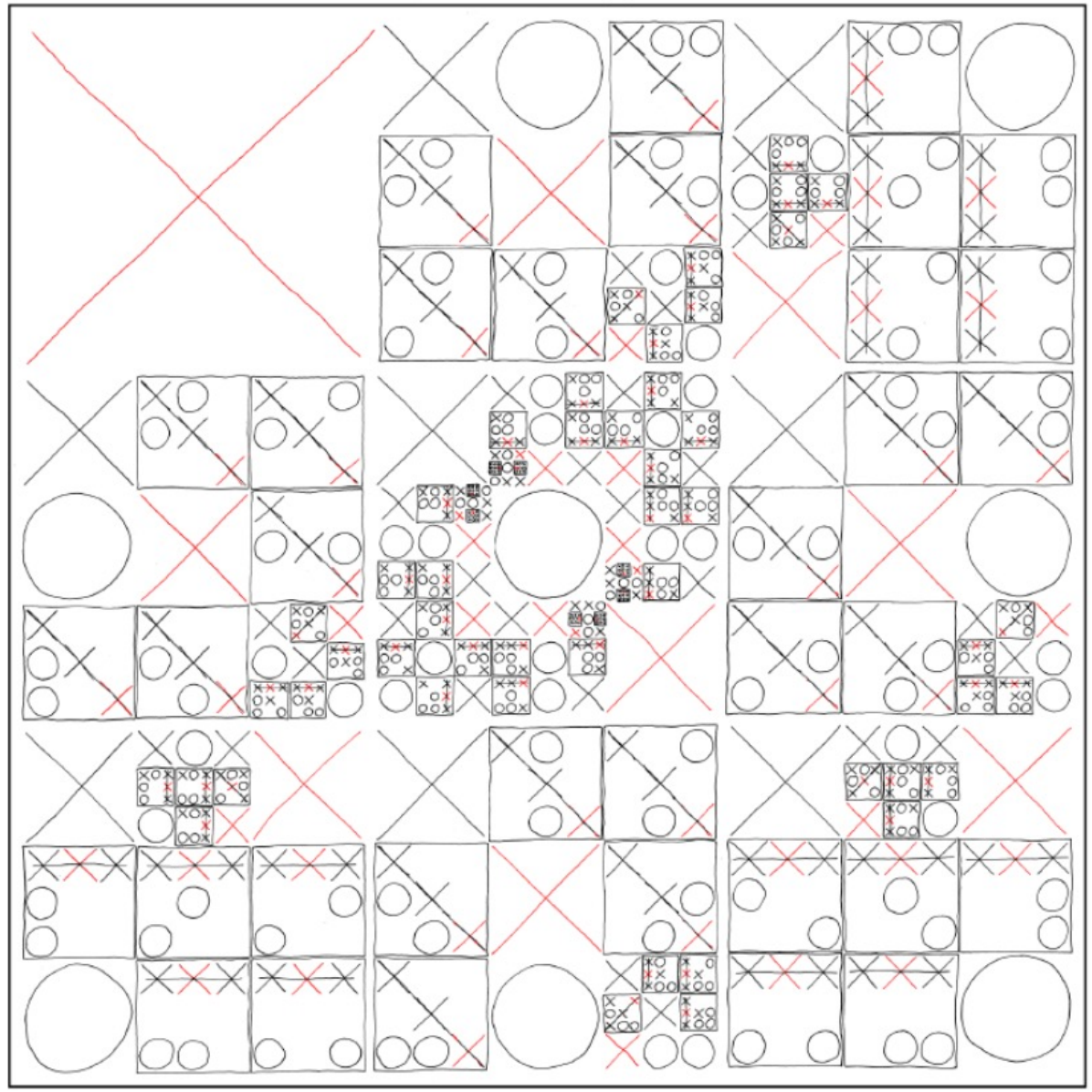
- X kann einen Sieg erzwingen
- muss dafür aber optimal spielen
- sonst gewinnt O

- Um zu erkennen, dass X sicher gewinnt, muss man 3 Züge in die Zukunft **planen**
- Die Zukunft zu antizipieren und zu beinflussen ist ein zentraler Aspekt von Intelligenz

COMPLETE MAP OF OPTIMAL TIC-TAC-TOE MOVES

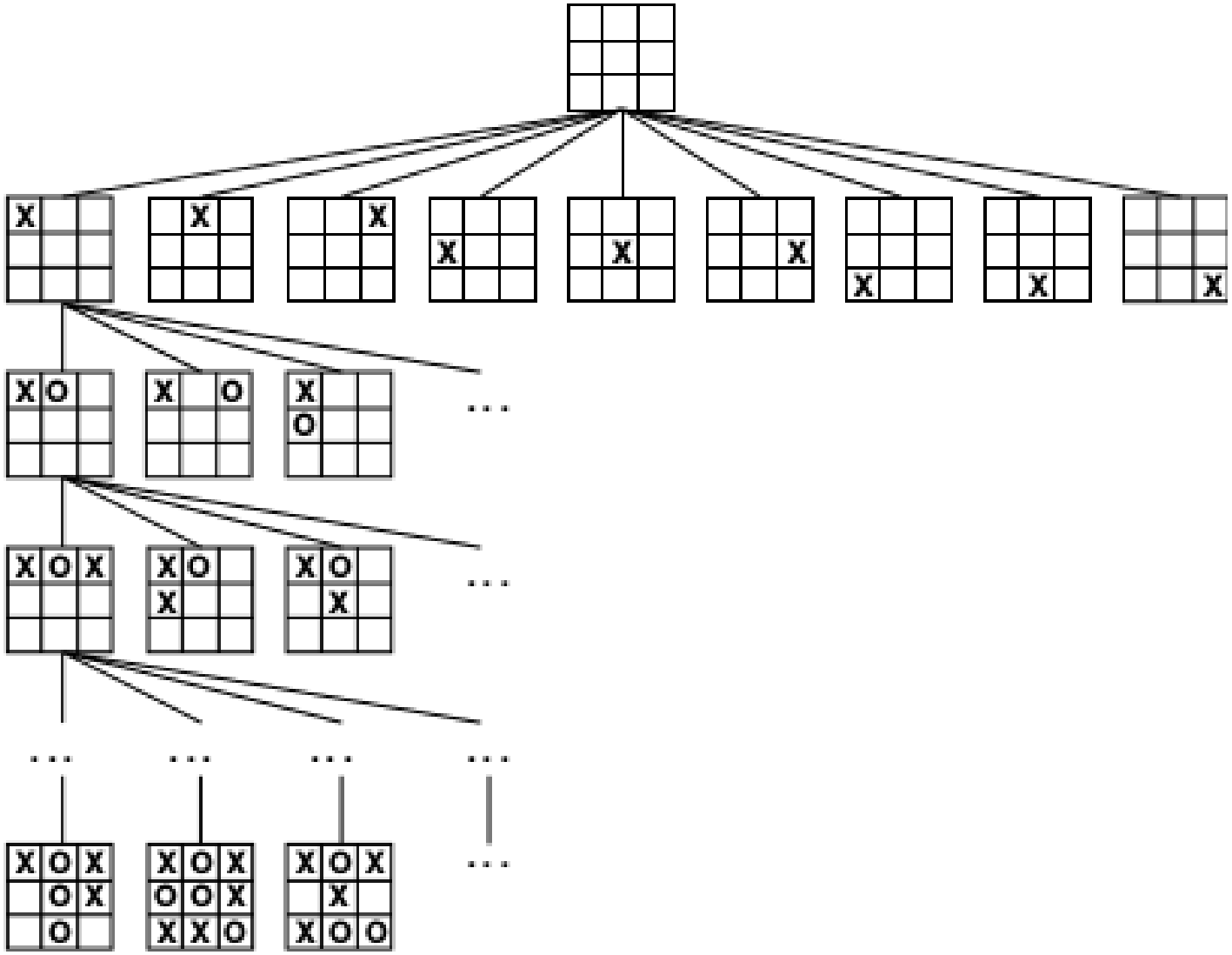
YOUR MOVE IS GIVEN BY THE POSITION OF THE LARGEST RED SYMBOL ON THE GRID. WHEN YOUR OPPONENT PICKS A MOVE, ZOOM IN ON THE REGION OF THE GRID WHERE THEY WENT. REPEAT.

MAP FOR X:



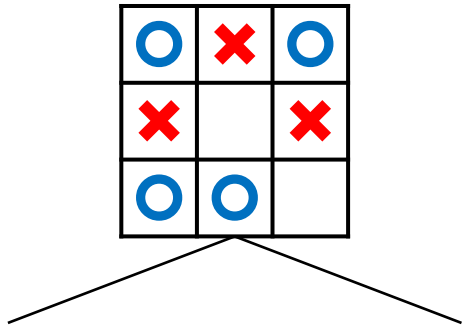
https://xkcd.com/832

Der Spielbaum von Tic Tac Toe (Ausschnitt!)



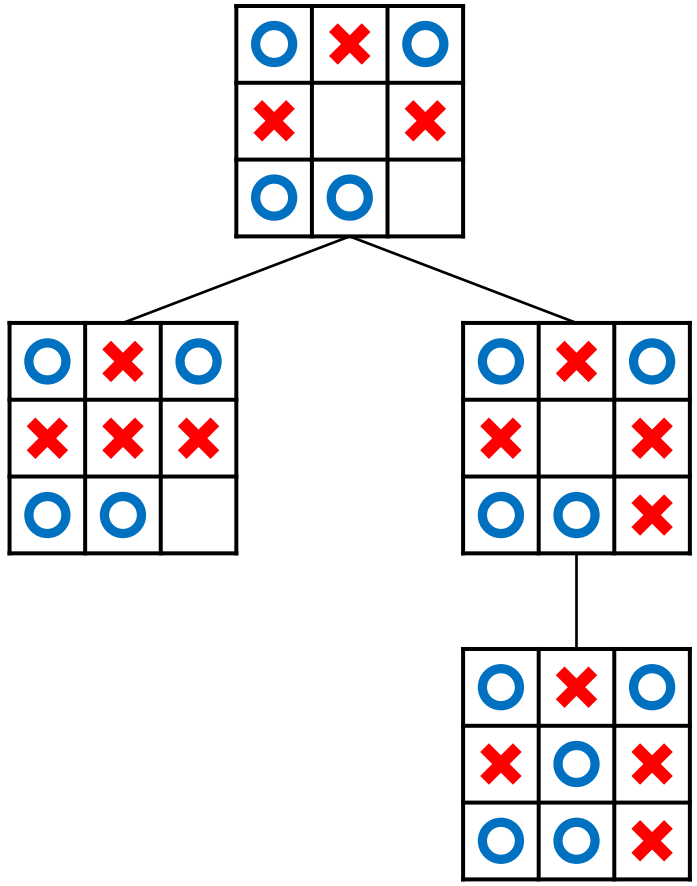
Vervollständige den Spielbaum.

(X ist am Zug)

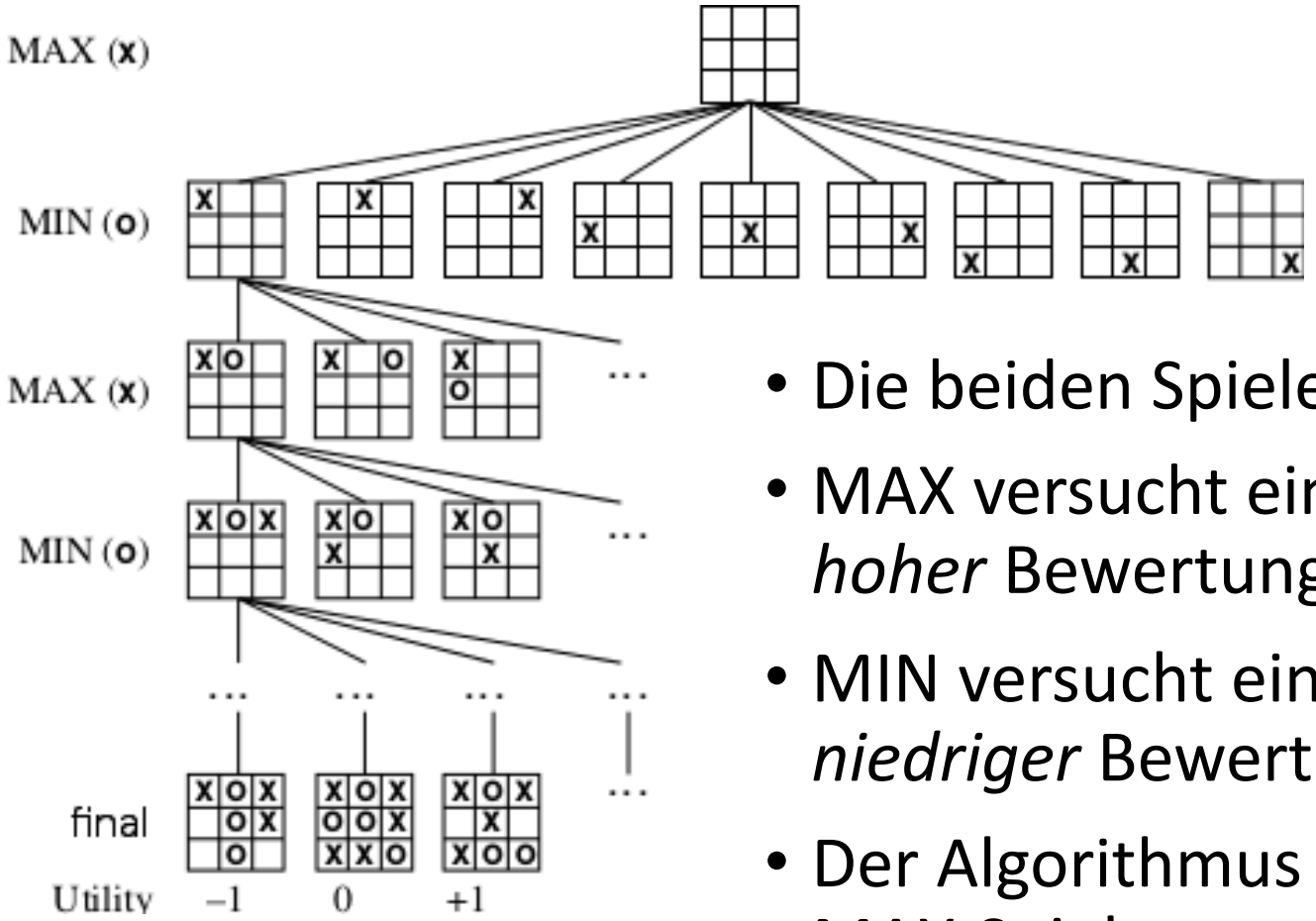


Vervollständige den Spielbaum.

(X ist am Zug)



Warum heißt es „Minimax“?

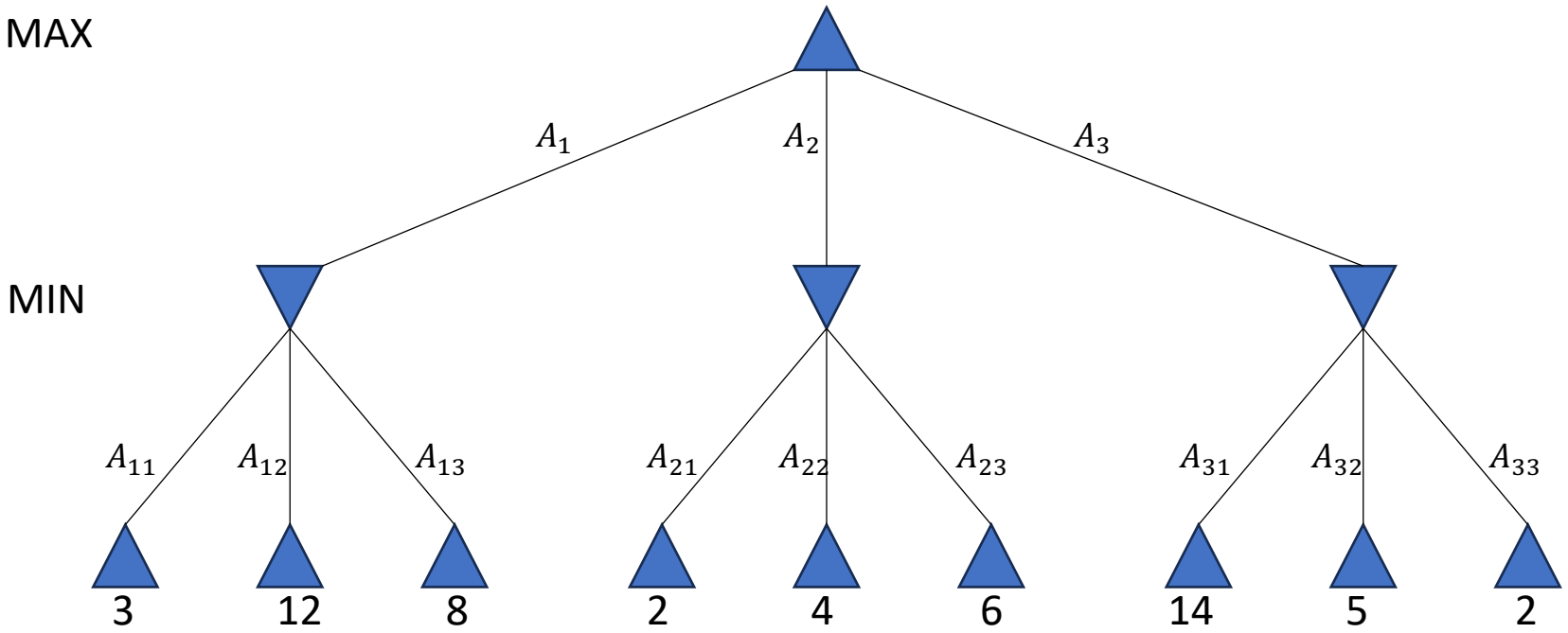


- Die beiden Spieler werden MAX und MIN genannt
- MAX versucht einen Zielzustand mit möglichst *hoher* Bewertung zu erreichen
- MIN versucht einen Zielzustand mit möglichst *niedriger* Bewertung zu
- Der Algorithmus betrachtet sich standardmäßig als MAX-Spieler

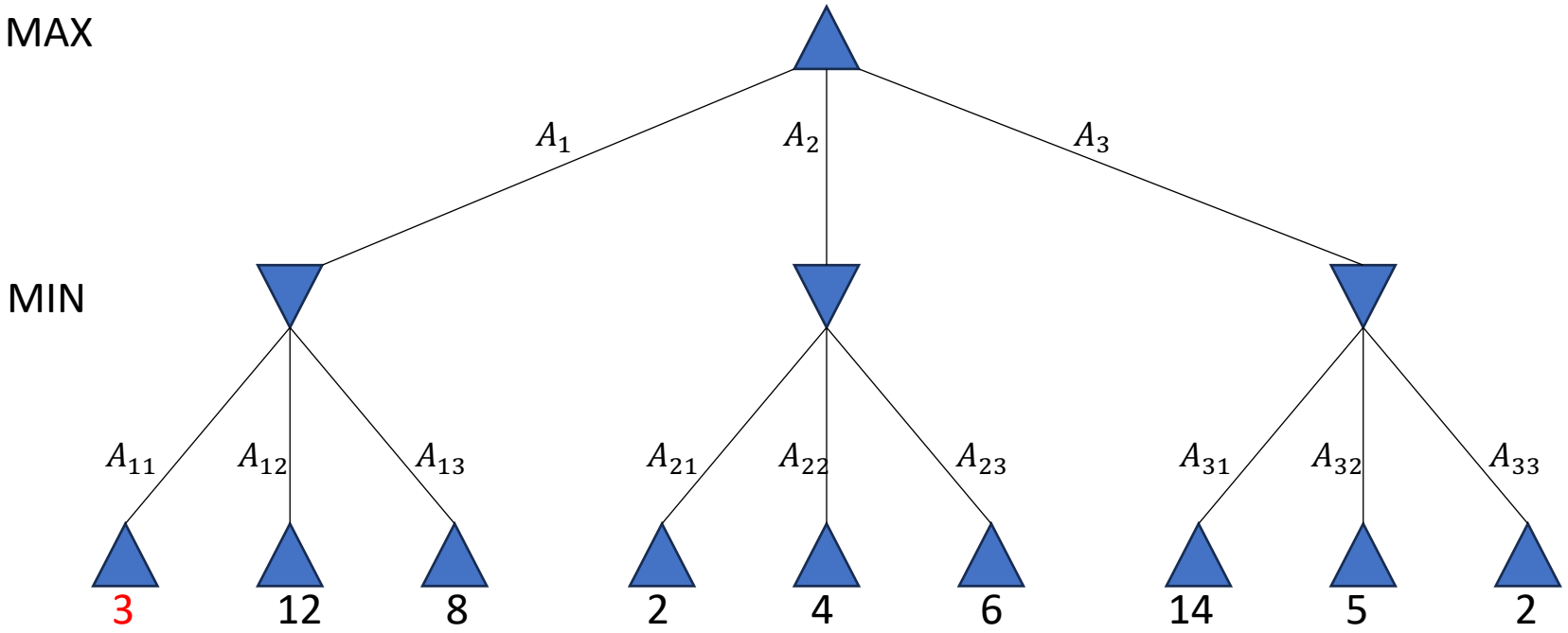
Der Minimax-Algorithmus

- Durchlaufe den Baum in Tiefensuche und berechne dabei jeweils den Wert dieses Zustands und den besten Folgezug
- Falls du in einem Endzustand *e* bist, ergibt sich der Wert direkt durch die Bewertungsfunktion als *b(e)*. Gib diesen Wert an den vorhergehenden Knoten zurück
- Falls du in keinem Endzustand bist, dann unterscheide:
  - MIN ist am Zug: Bewertung ist das Minimum der Bewertungen der Nachfolgezustände
  - MAX ist am Zug: Bewertung ist das Maximum der Bewertungen der Nachfolgezustände
- Im Wurzelknoten: MAX wählt den Zug, der zur maximalen Bewertung führt

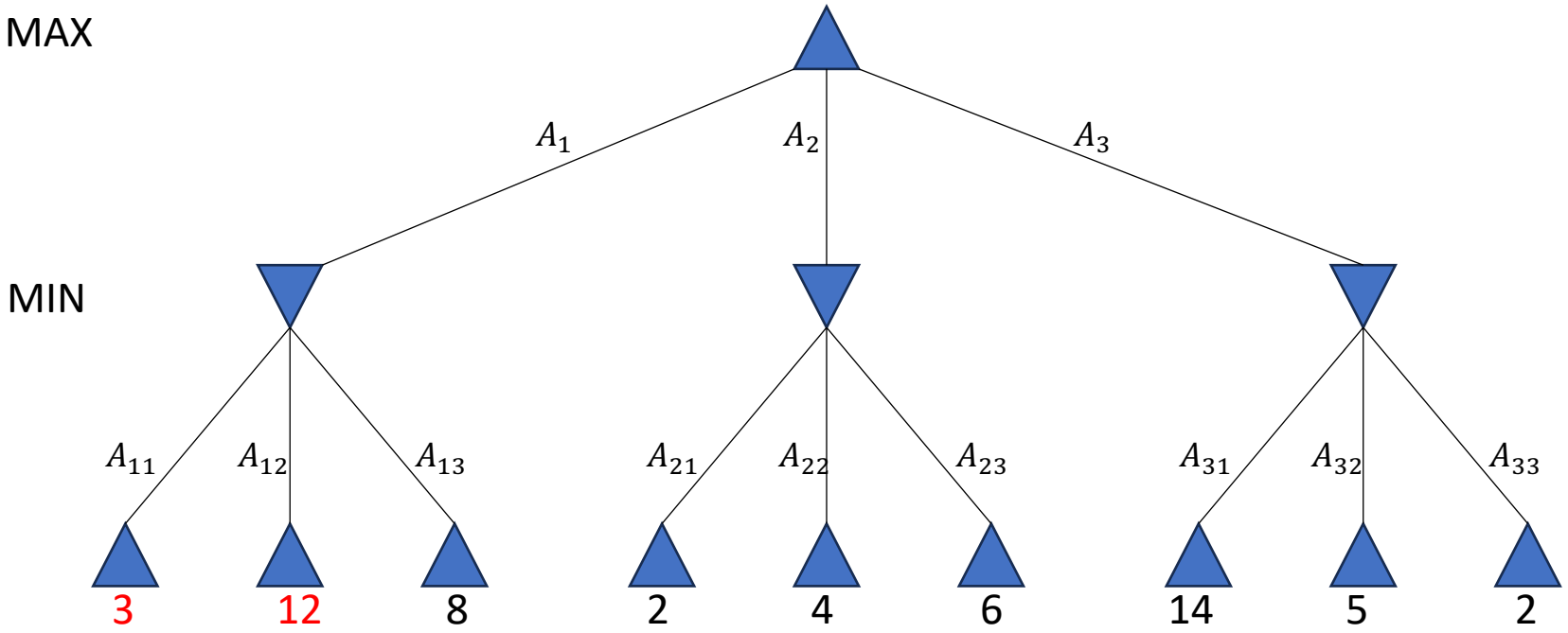
Beispiel: Welchen Zug sollte MAX wählen?



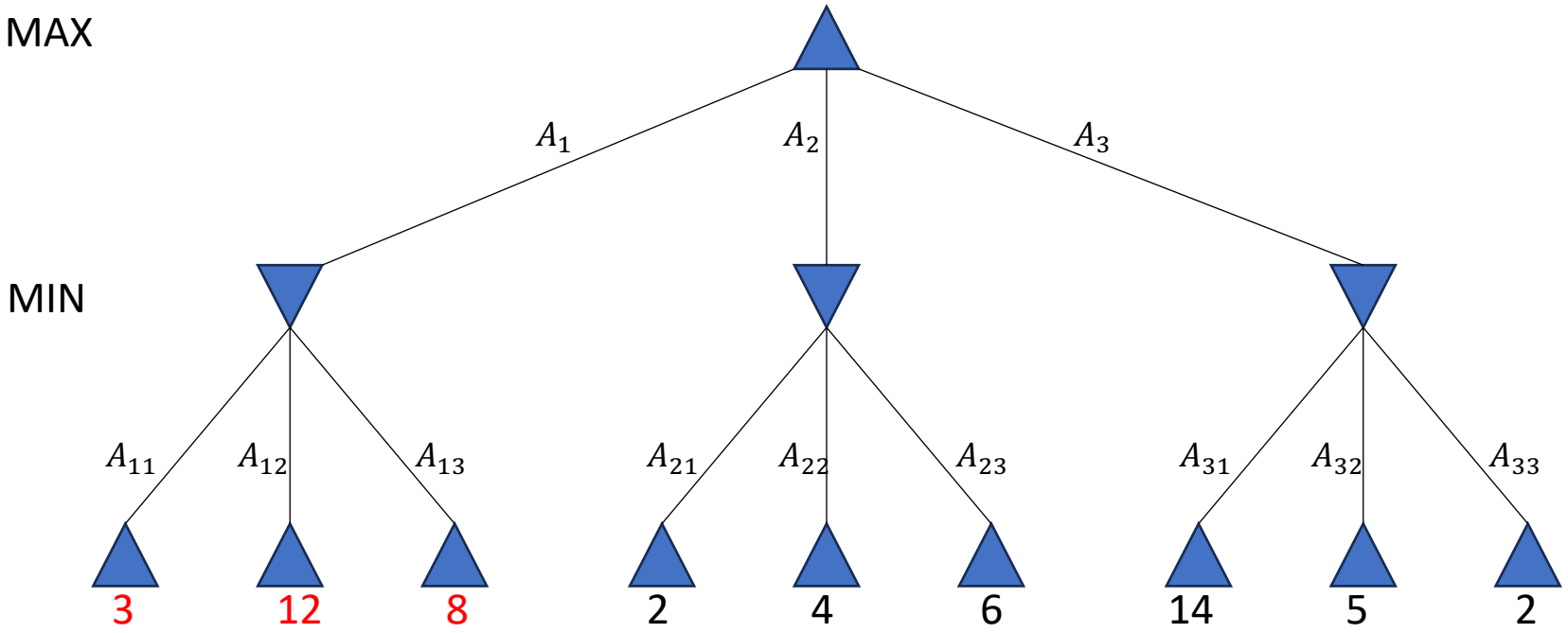
Beispiel


MAX
MIN
A_1
A_2
A_3
A_{11}
A_{12}
A_{13}
A_{21}
A_{22}
A_{23}
A_{31}
A_{32}
A_{33}
3
12
8
2
4
6
14
5
2

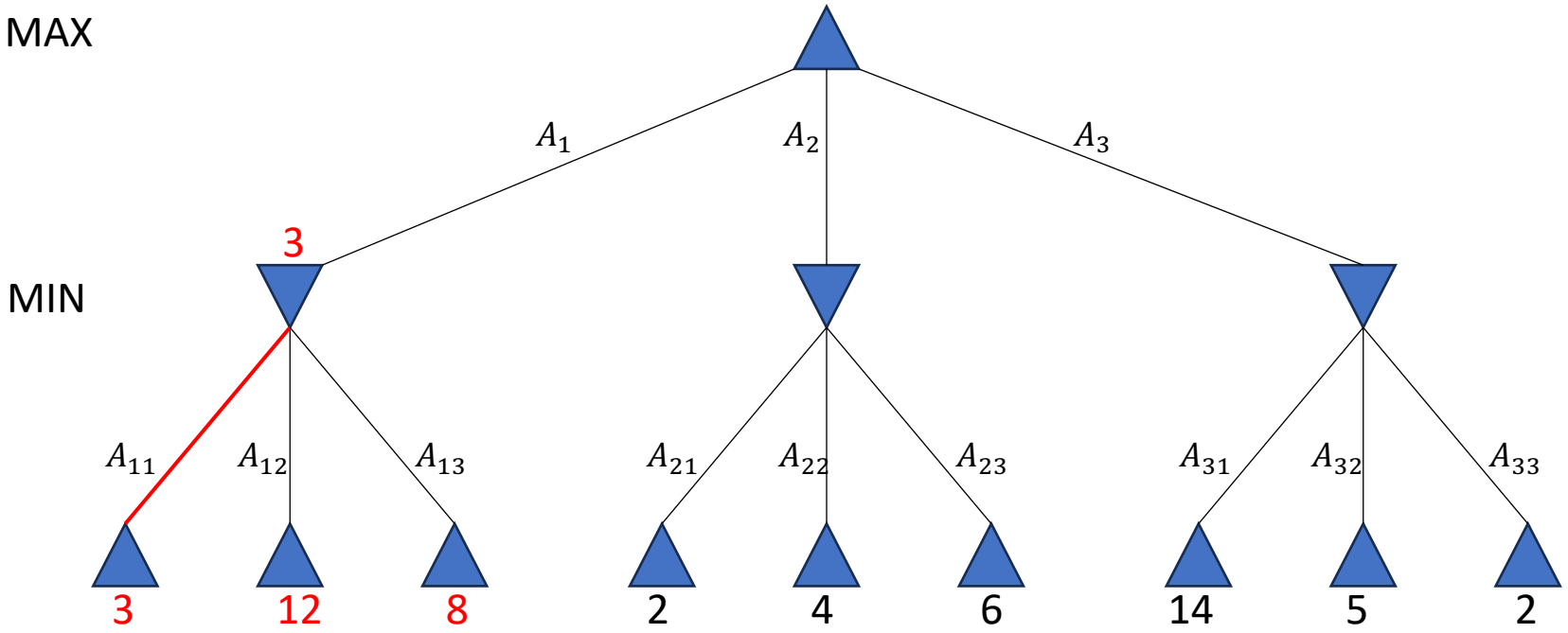
Beispiel


MAX
MIN
A_1
A_2
A_3
A_{11}
A_{12}
A_{13}
A_{21}
A_{22}
A_{23}
A_{31}
A_{32}
A_{33}
3
12
8
2
4
6
14
5
2

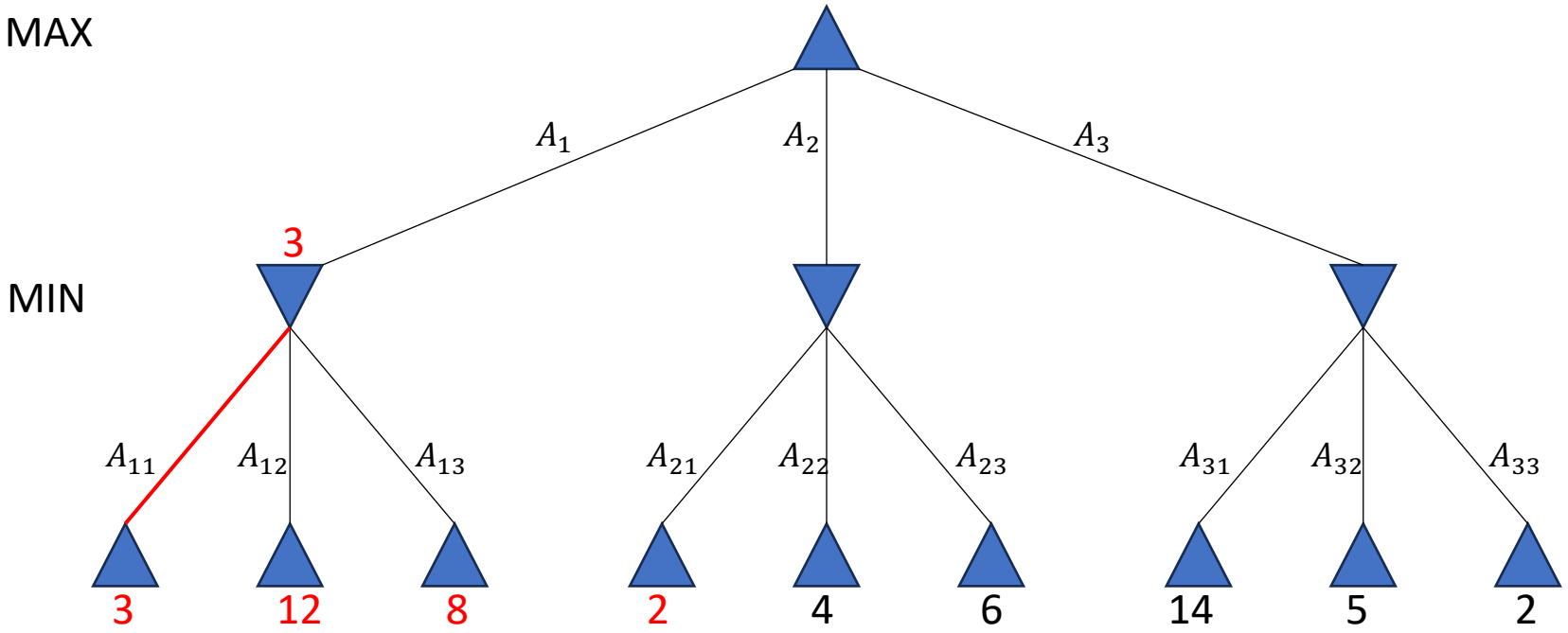
Beispiel


MAX
MIN
A_1
A_2
A_3
A_{11}
A_{12}
A_{13}
A_{21}
A_{22}
A_{23}
A_{31}
A_{32}
A_{33}
3
12
8
2
4
6
14
5
2

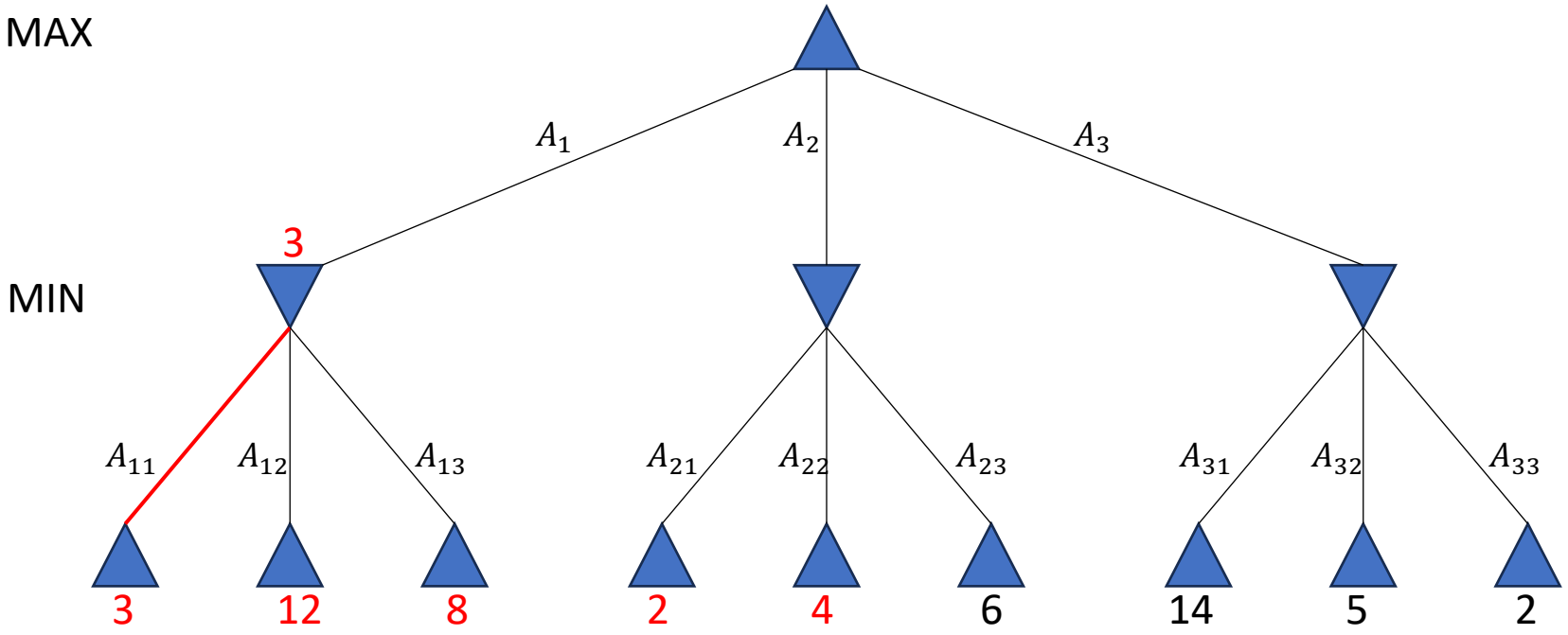
Beispiel


MAX
MIN
A_1
A_2
A_3
3
A_{11}
A_{12}
A_{13}
A_{21}
A_{22}
A_{23}
A_{31}
A_{32}
A_{33}
3
12
8
2
4
6
14
5
2

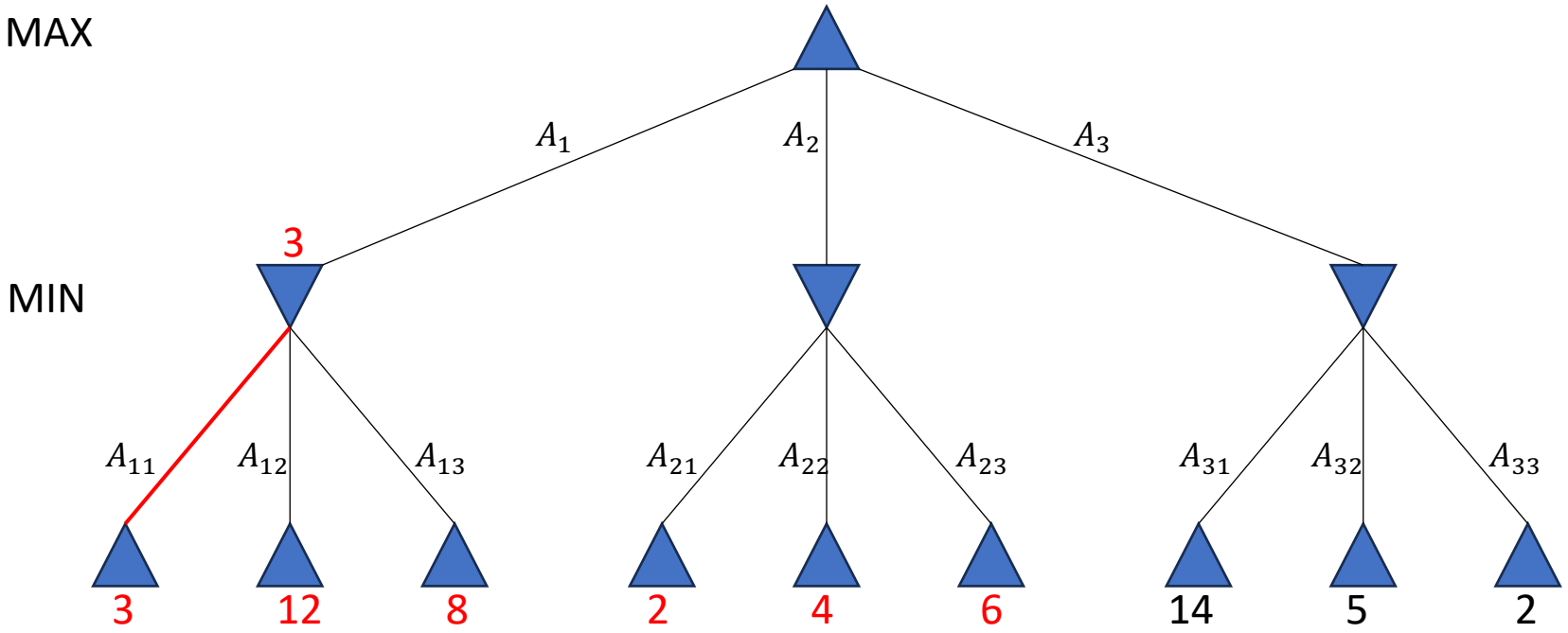
Beispiel


MAX
MIN
A_1
A_2
A_3
3
A_{11}
A_{12}
A_{13}
A_{21}
A_{22}
A_{23}
A_{31}
A_{32}
A_{33}
3
12
8
2
4
6
14
5
2

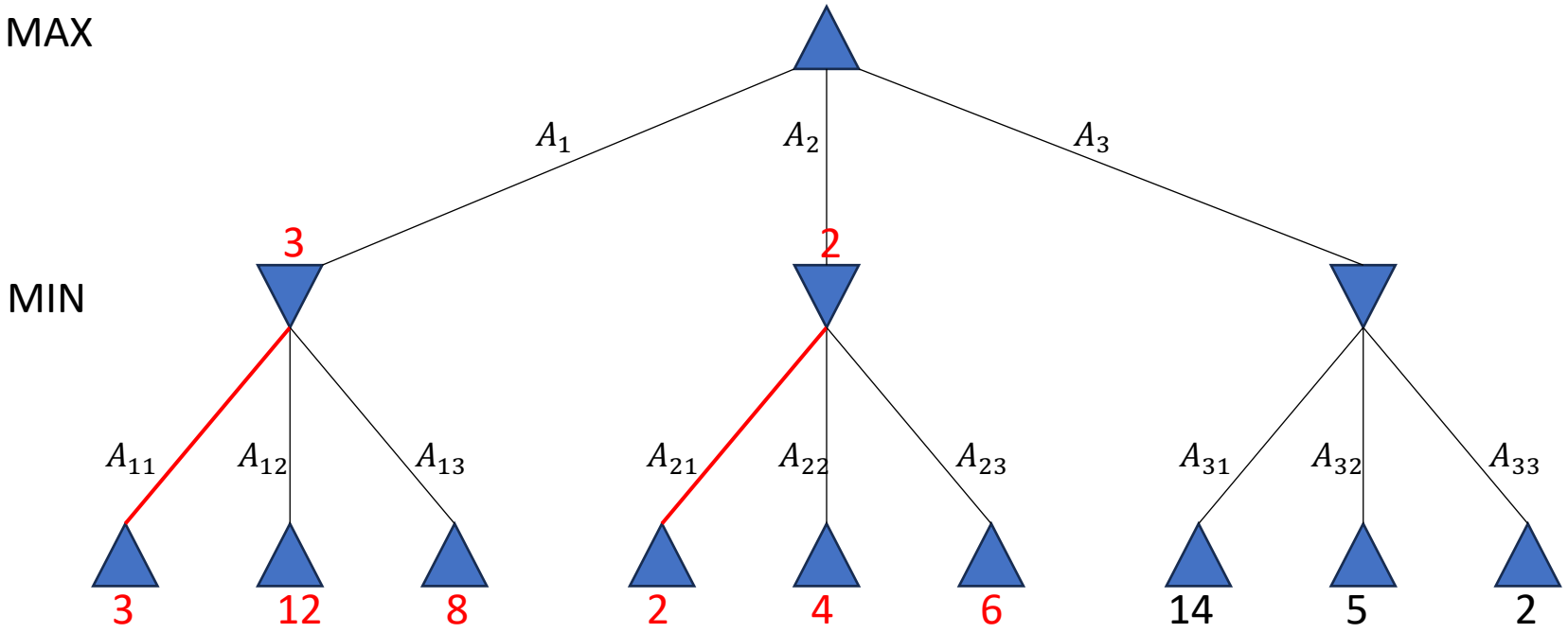
Beispiel


MAX
MIN
A_1
A_2
A_3
3
A_{11}
A_{12}
A_{13}
A_{21}
A_{22}
A_{23}
A_{31}
A_{32}
A_{33}
3
12
8
2
4
6
14
5
2

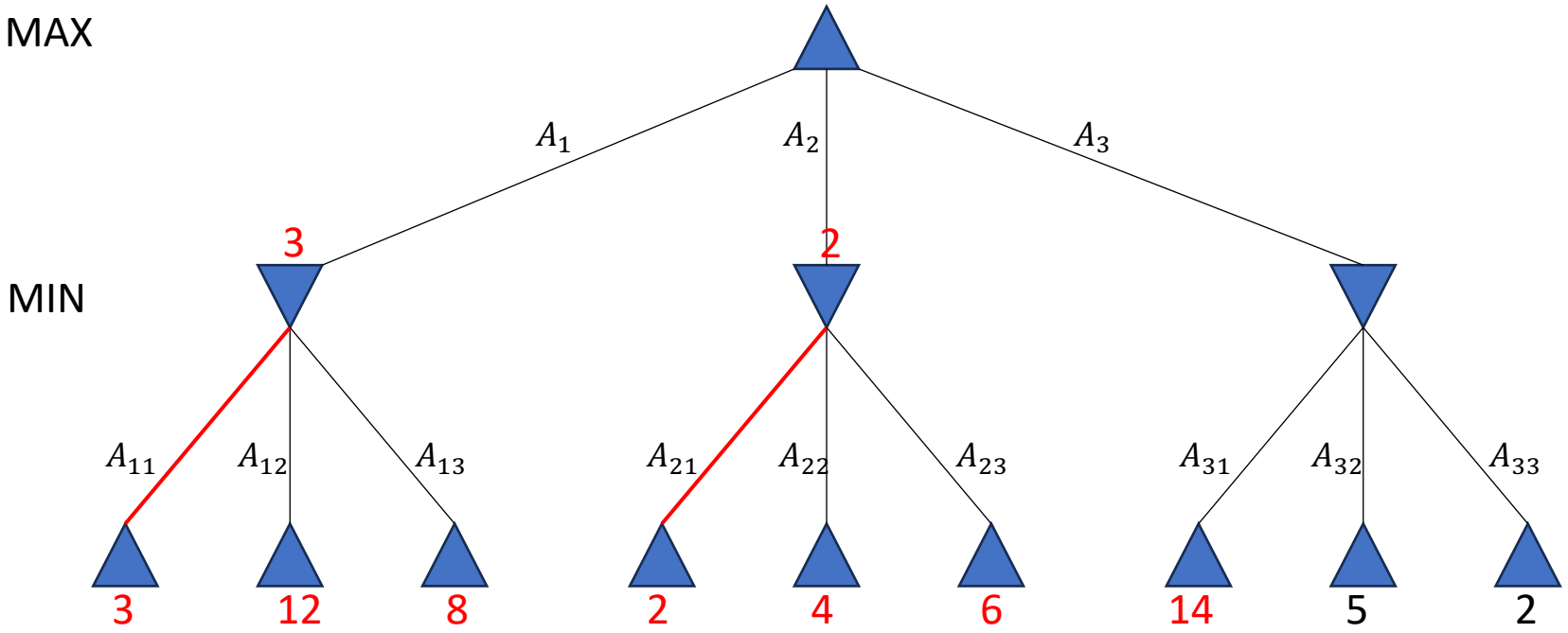
Beispiel


MAX
MIN
A_1
A_2
A_3
3
A_{11}
A_{12}
A_{13}
A_{21}
A_{22}
A_{23}
A_{31}
A_{32}
A_{33}
3
12
8
2
4
6
14
5
2

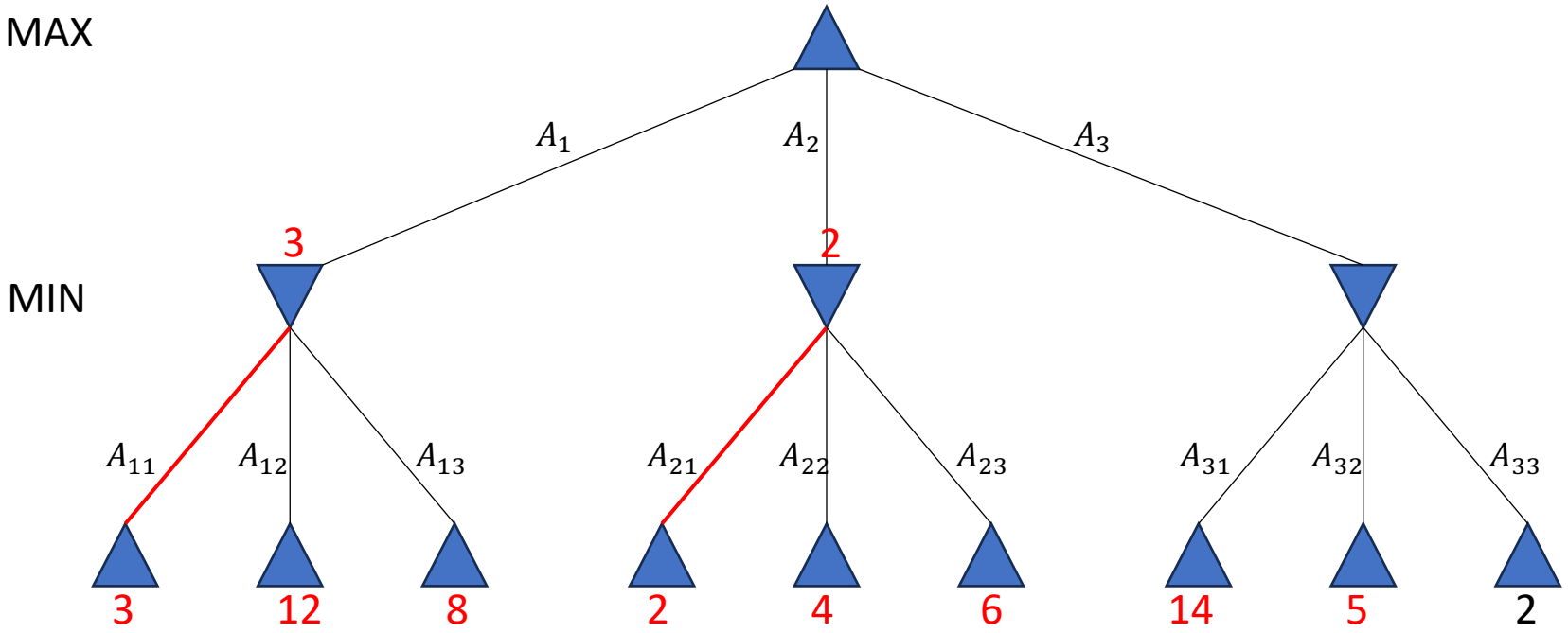
Beispiel


MAX
MIN
A_1
A_2
A_3
3
2
A_{11}
A_{12}
A_{13}
A_{21}
A_{22}
A_{23}
A_{31}
A_{32}
A_{33}
3
12
8
2
4
6
14
5
2

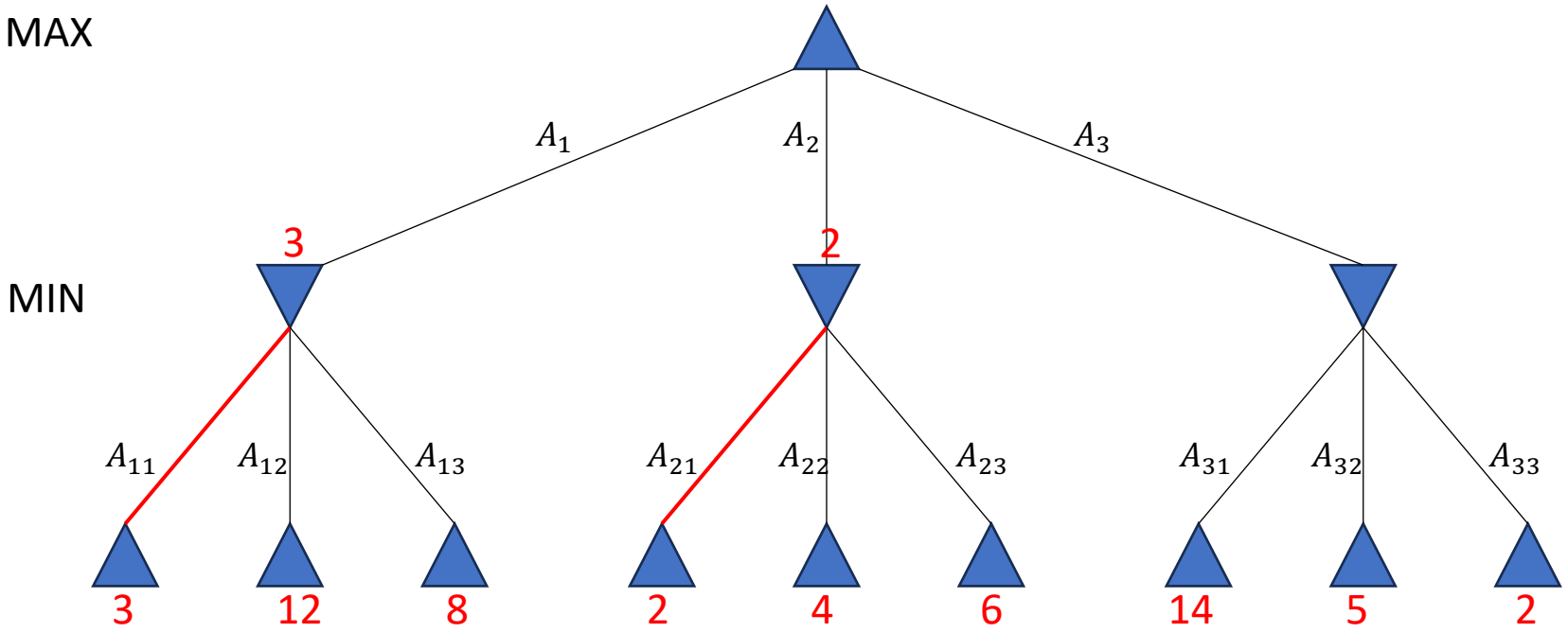
Beispiel


MAX
MIN
A_1
A_2
A_3
3
2
A_{11}
A_{12}
A_{13}
A_{21}
A_{22}
A_{23}
A_{31}
A_{32}
A_{33}
3
12
8
2
4
6
14
5
2

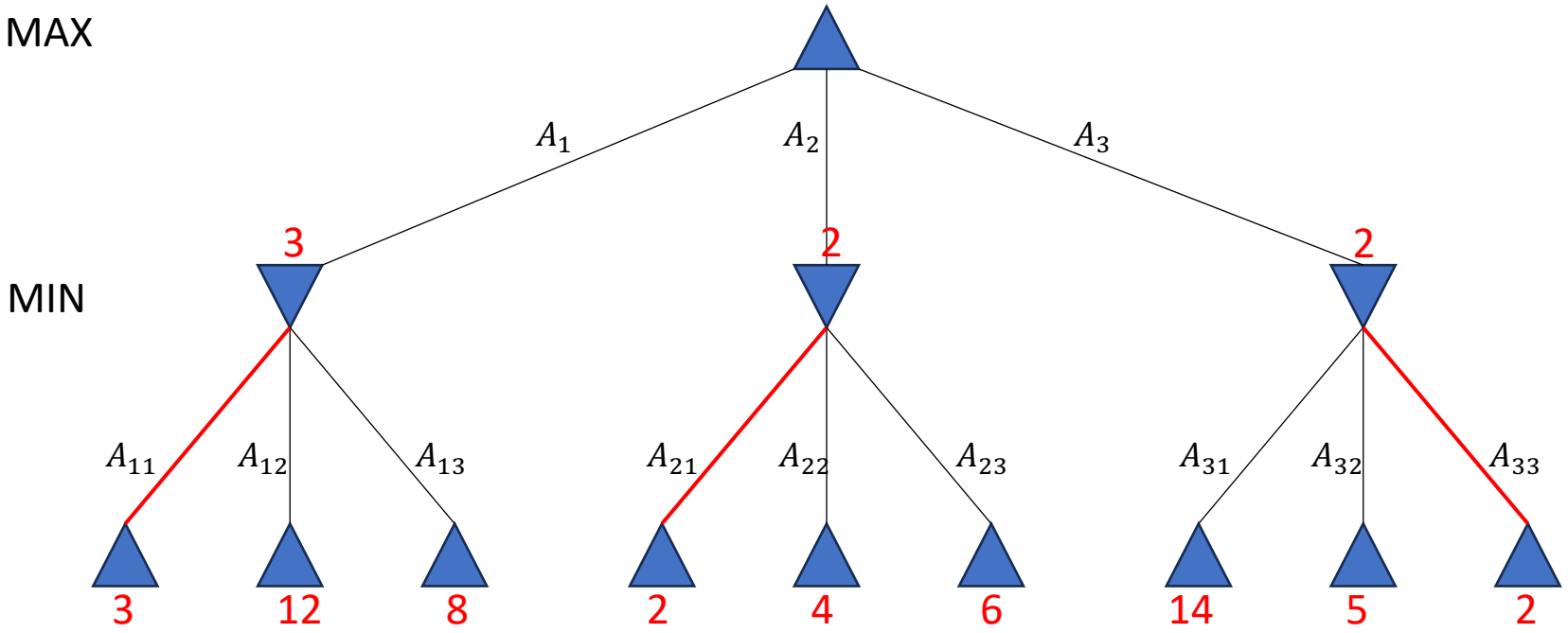
Beispiel


MAX
MIN
A_1
A_2
A_3
3
2
A_{11}
A_{12}
A_{13}
A_{21}
A_{22}
A_{23}
A_{31}
A_{32}
A_{33}
3
12
8
2
4
6
14
5
2

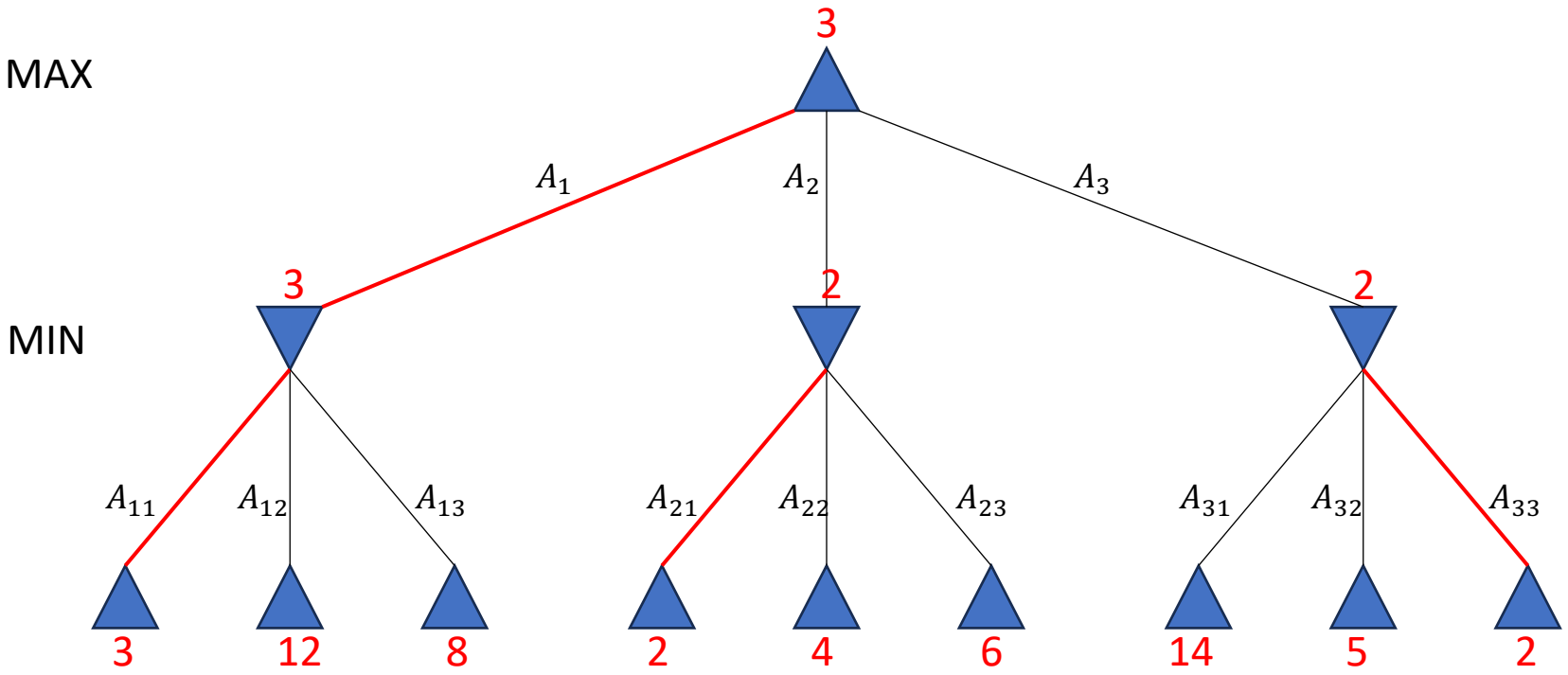
Beispiel


MAX
MIN
A_1
A_2
A_3
3
2
A_{11}
A_{12}
A_{13}
A_{21}
A_{22}
A_{23}
A_{31}
A_{32}
A_{33}
3
12
8
2
4
6
14
5
2

Beispiel


MAX
MIN
A_1
A_2
A_3
3
2
2
A_{11}
A_{12}
A_{13}
A_{21}
A_{22}
A_{23}
A_{31}
A_{32}
A_{33}
3
12
8
2
4
6
14
5
2

Beispiel


MAX
MIN
3
A_1
A_2
A_3
3
2
2
A_{11}
A_{12}
A_{13}
A_{21}
A_{22}
A_{23}
A_{31}
A_{32}
A_{33}
3
12
8
2
4
6
14
5
2

Minimax in Pseudocode

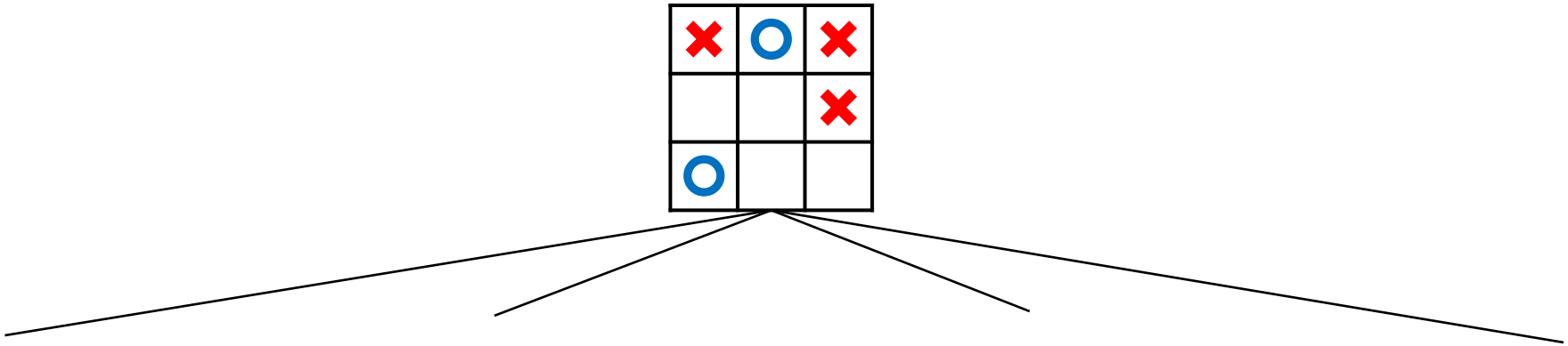
function minimax(p)

if p is terminal position:
    return $\langle u(p), \textbf{none} \rangle$
$best\_move := \textbf{none}$
if $player(p) = \text{MAX}$:
    $v := -\infty$
else:
    $v := \infty$
for each $\langle move, p' \rangle \in \text{succ}(p)$:
    $\langle v', best\_move' \rangle := minimax(p')$
    if ($player(p) = \text{MAX}$ **and** $v' > v$) **or**
       ($player(p) = \text{MIN}$ **and** $v' < v$):
        $v := v'$
        $best\_move := move$
return $\langle v, best\_move \rangle$

Minimax in Python

```python
def minimax(spielfeld: Zustand, ist_maximierer: bool) -> tuple[Zug, int]:

    if spielfeld.ist_endzustand():
        # Basisfall: Spiel ist zu Ende, daher keine weiteren Züge möglich
        return DUMMY_ZUG, spielfeld.bewertung()

    bester_zug = DUMMY_ZUG  # Initialisierung mit Dummy-Zug und Dummy-Wert
    bester_wert = -UNENDLICH if ist_maximierer else UNENDLICH

    for zug in spielfeld.moegliche_zuege():
        neues_spielfeld = spielfeld.führe_zug_aus(zug)
        _, wert = minimax(neues_spielfeld, ist_maximierer=not ist_maximierer)
        if (ist_maximierer == True and wert > bester_wert) or (
            ist_maximierer == False and wert < bester_wert):
            bester_zug = zug # besserer Zug gefunden
            bester_wert = wert

    return bester_zug, bester_wert
```

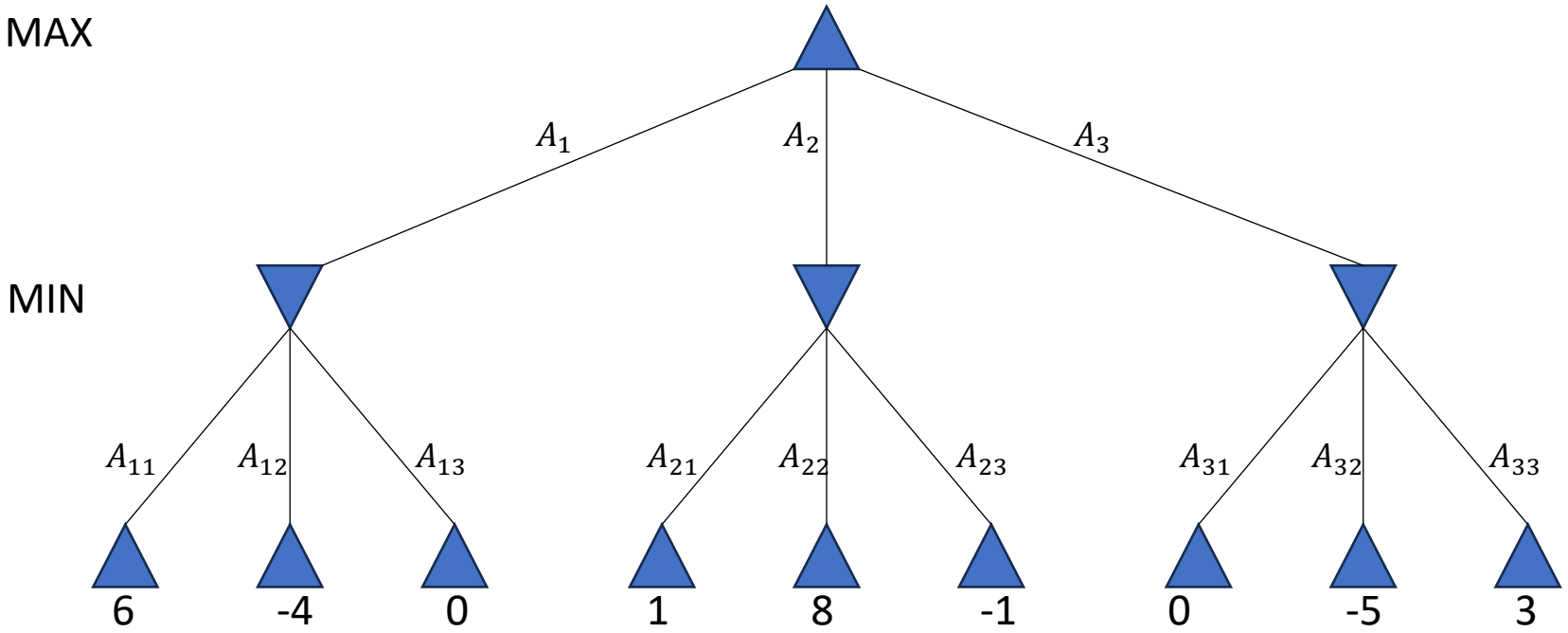
O ist am Zug. Vervollständige den Spielbaum, bewerte den aktuellen Zustand mit dem Minimax-Algorithmus.



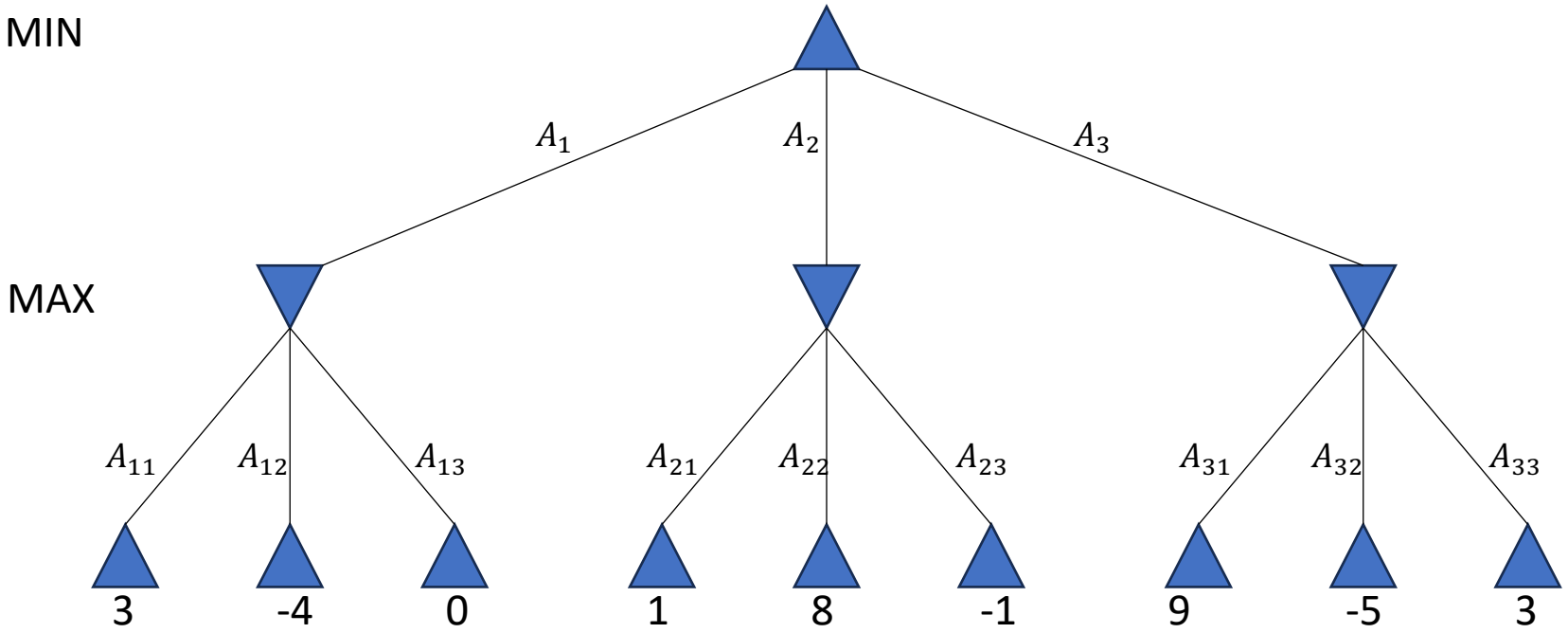
Was tun, wenn der Spielbaum zu groß ist?

- Minimax besucht *alle* Blätter des Spielbaums → exponentiell viele Spielzustände
- Idee: Echte Bewertungsfunktion b approximieren durch Evaluationsfunktion e, die die „Güte“ eines Zustands abschätzt, auch wenn dieser noch kein Endzustand ist
- Beispiel Schach:
  - Figuren haben einen Wert (Dame = 9, Turm = 5, …, Bauer = 1)
  - Zustandsevaluation: Summe Figuren weiß – Summe Figuren schwarz

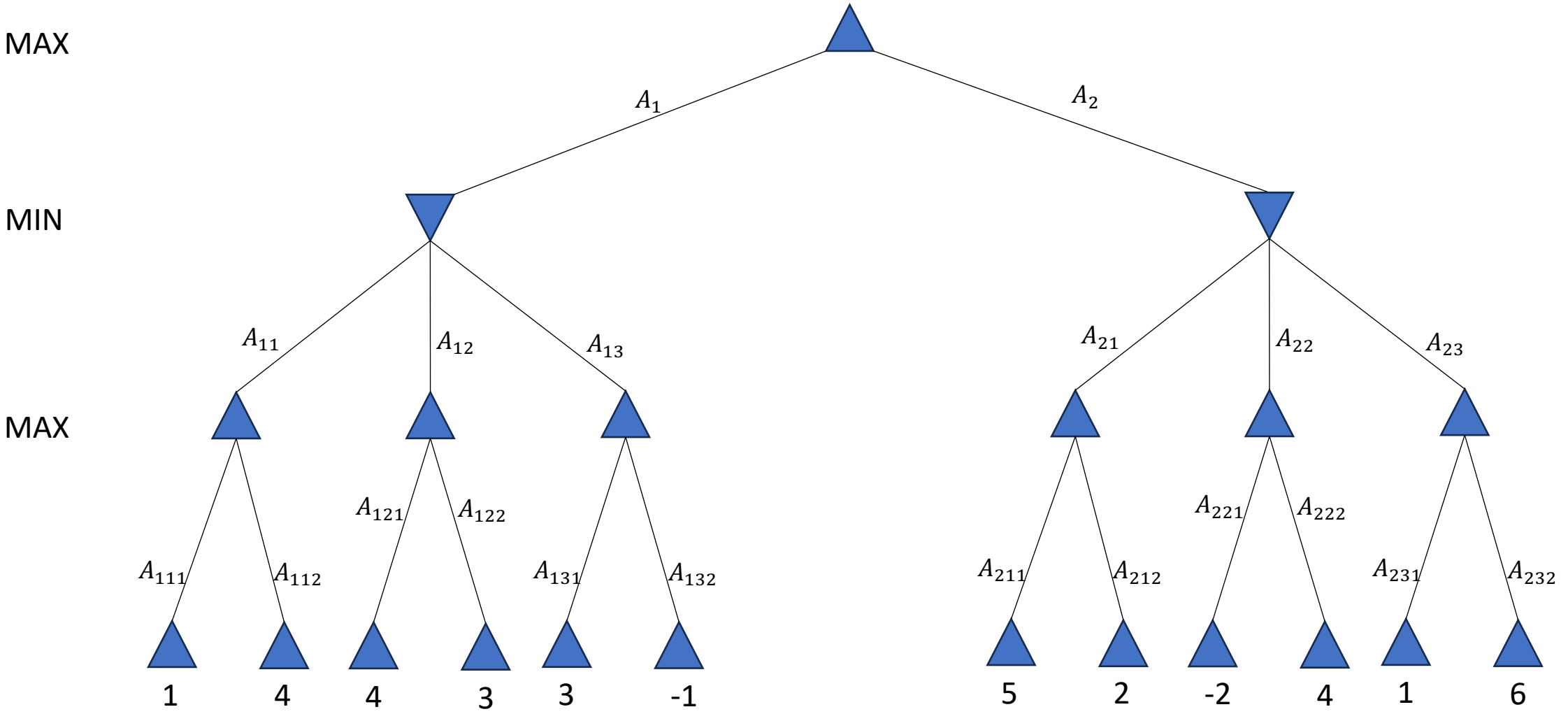
Aufgabe: Welchen Zug sollte MAX wählen und wie viele Punkte kann er erreichen?



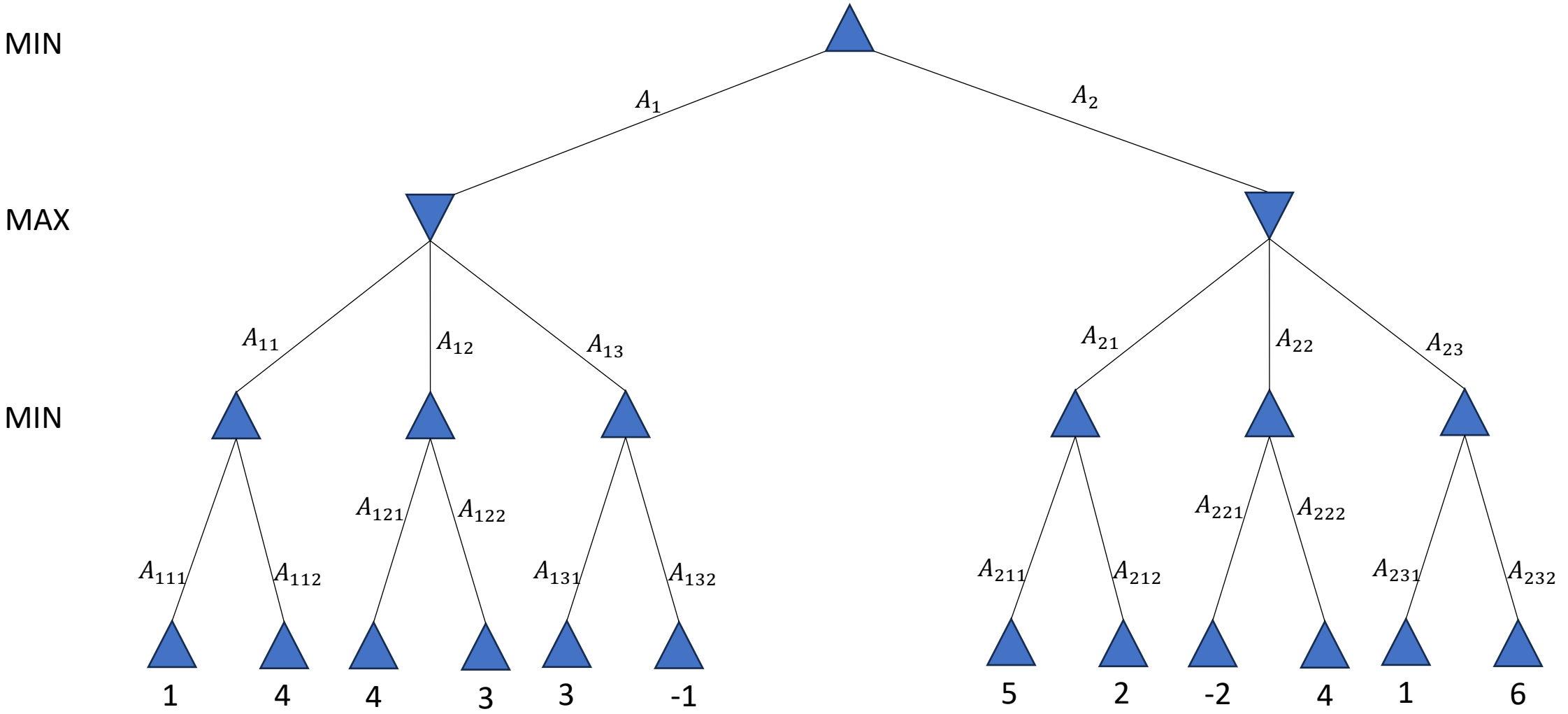
Aufgabe: Welchen Zug sollte MIN wählen und wie viele Punkte kann er erreichen?



Aufgabe: Welchen Zug sollte MAX wählen und wie viele Punkte kann er erreichen?



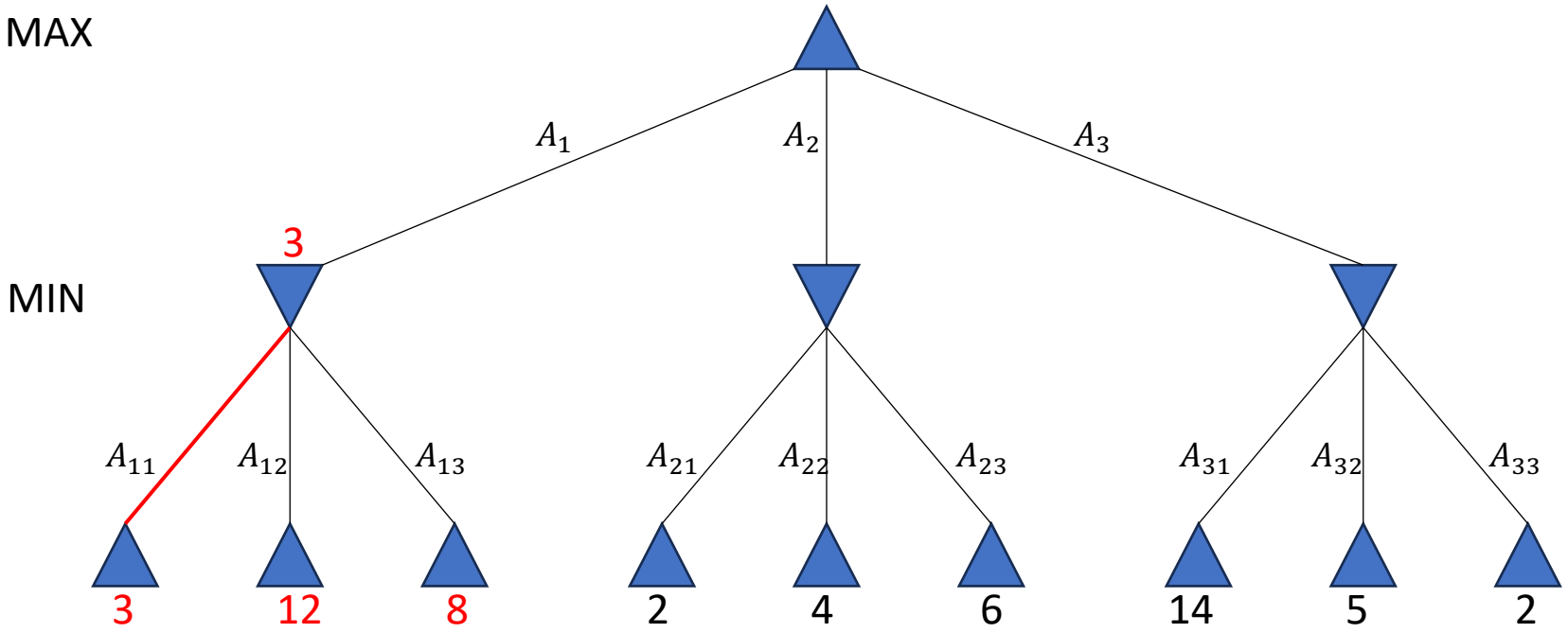
Aufgabe: Welchen Zug sollte MIN wählen und wie viele Punkte kann er erreichen?



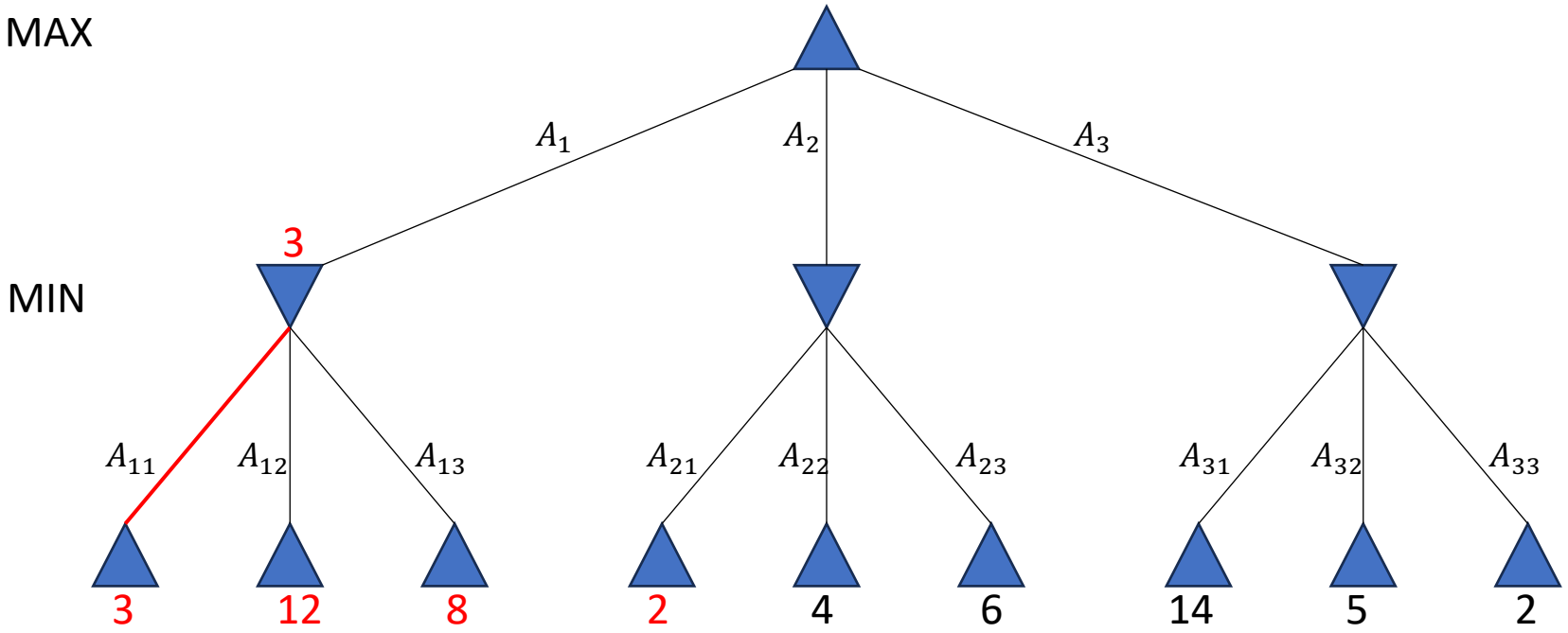
Verbesserung durch Alpha-Beta-Pruning

- Minimax besucht *alle* Blätter des Spielbaums
  → exponentiell viele Spielzustände
- Wenn wir eine Bewertungsfunktion nutzen wird das Ergebnis ungenauer → besser erst möglich tief im Suchbaum bewerten
- also will man trotzdem möglichst große Teile des exponentiell wachsenden Baums untersuchen
- Erkenntnis: Wenn ein (rationaler) Spieler bestimmte Züge nie machen wird, weil sie für ihn garantiert zu keiner besseren Bewertung führen → dann können wir diesen Teil des Spielbaums einfach *ignorieren*
- Alpha-Beta-Pruning

Alpha-Beta-Pruning



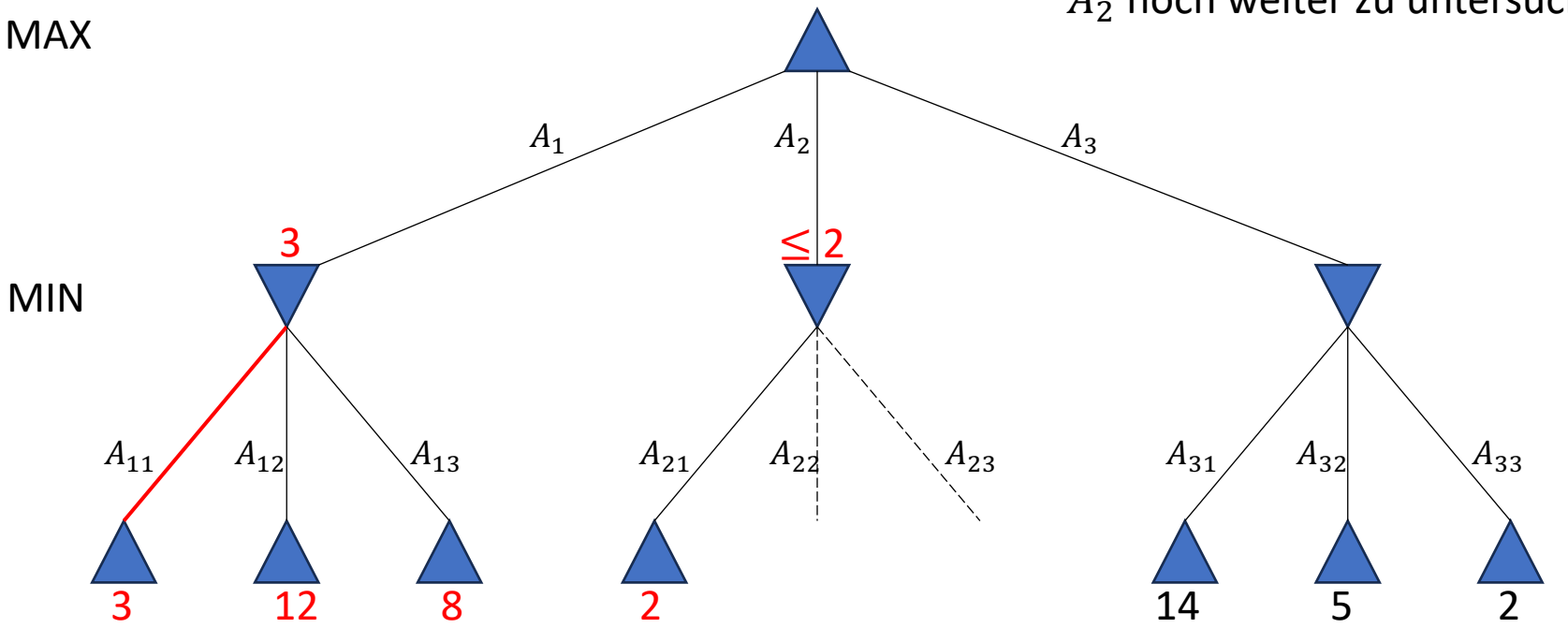
Alpha-Beta-Pruning



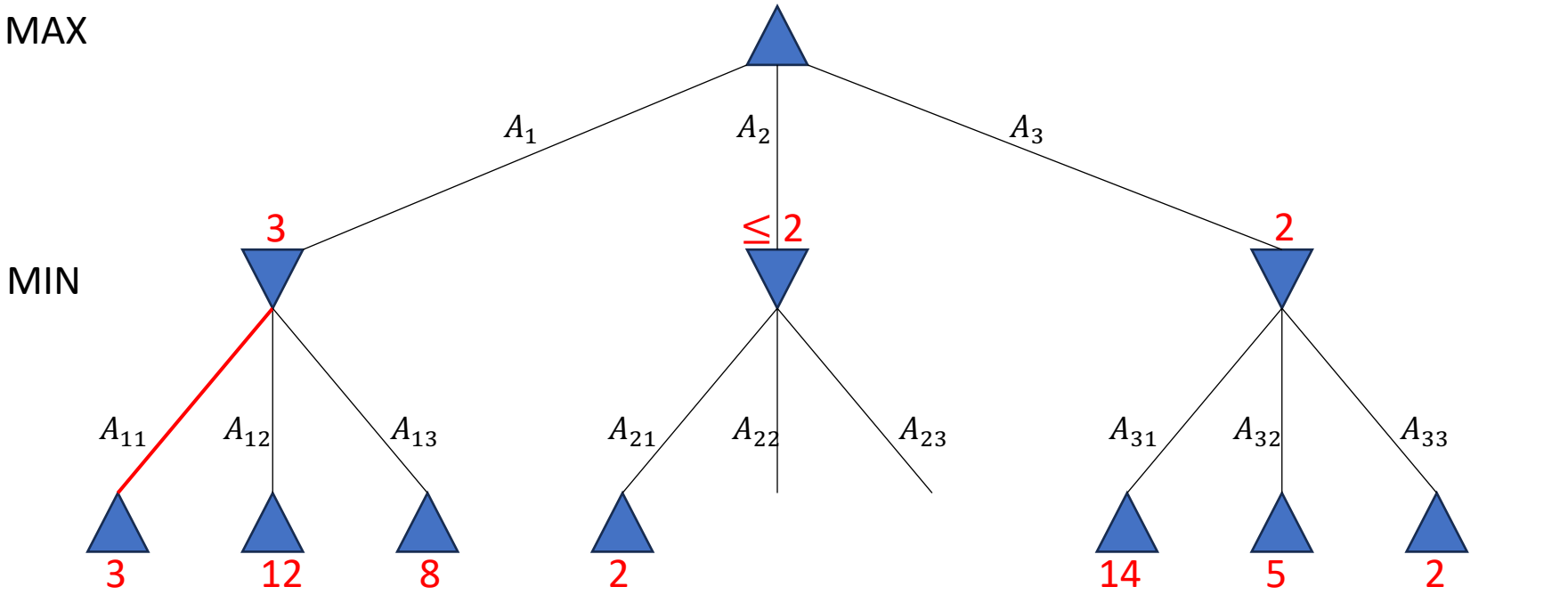
Alpha-Beta-Pruning

Sobald wir diese 2 sehen, wissen wir

1. dass MIN durch den Zug A_{21} mind. 2 oder besser (also noch weniger) garantieren kann
2. dass MAX ja aber schon 3 garantieren kann (durch A_1)
3. und deshalb auf keinen Fall A_2 spielen sollte
4. Es deshalb auch sinnlos ist, den Teilbaum unter A_2 noch weiter zu untersuchen

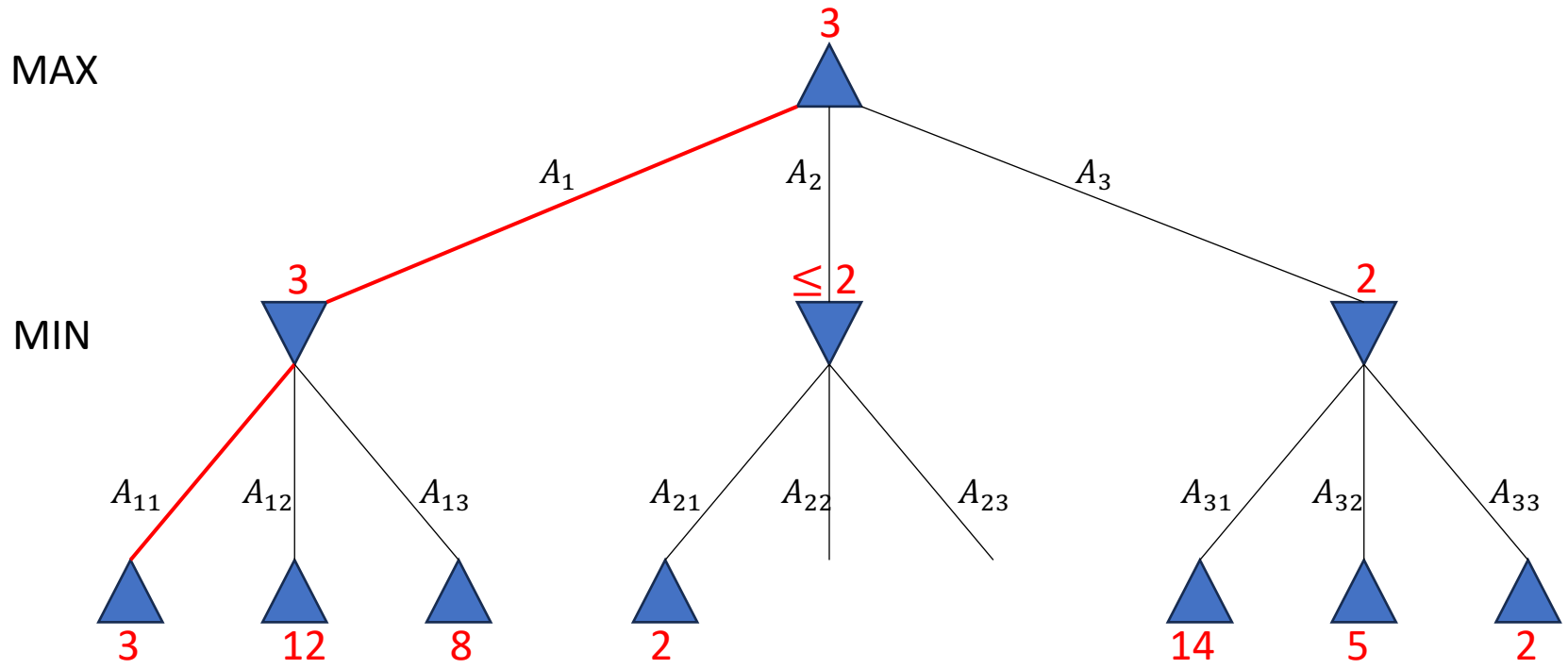


Alpha-Beta-Pruning



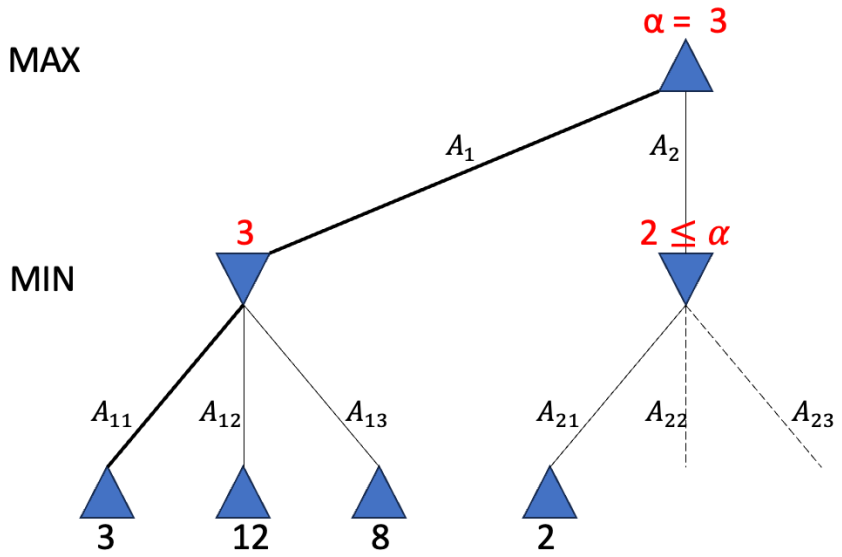
Aktionen in ungünstiger Reihenfolge betrachtet
→ kein Pruning möglich 😢

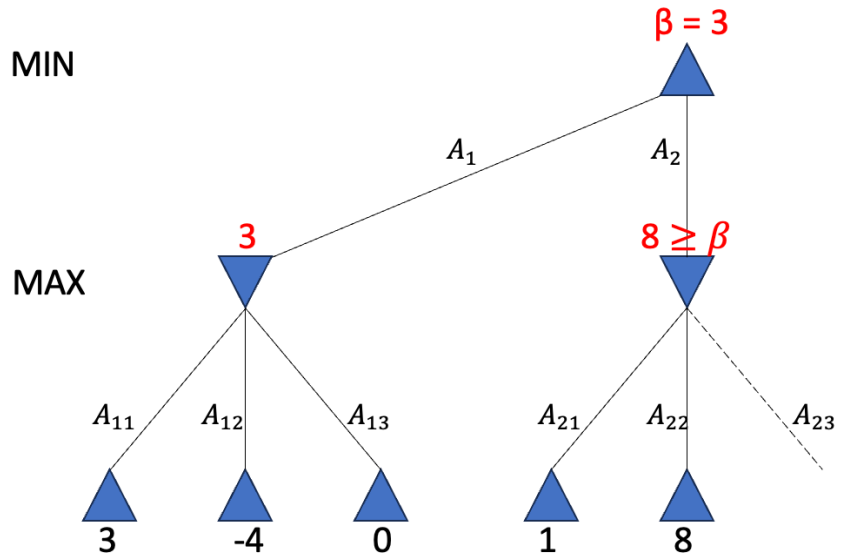
Alpha-Beta-Pruning



Alpha-Beta-Pruning formal

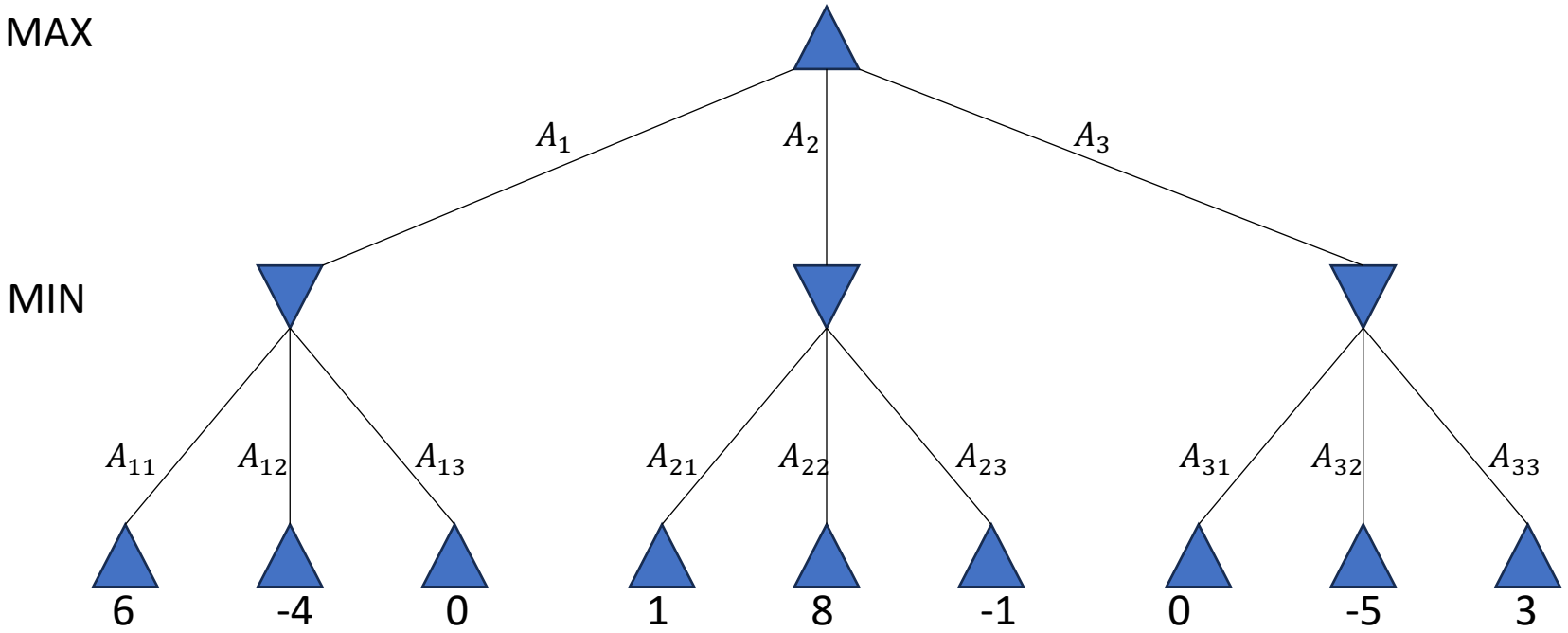
- **α:** Der (bisher) *höchste* Wert, den MAX garantiert erreichen kann.
- **β**: Der (bisher) *niedrigste* Wert, den MIN garantiert erreichen kann.
- Wenn ein MIN-Knoten einen Zug machen kann, der v Punkte garantiert und $v \leq \alpha$ ist, dann ist dieser Zug für den darüberliegenden MAX-Knoten schlechter als der vorher gefundene, der ihm α Punkte sichert.
- Die noch nicht betrachteten MIN-Züge können es für MAX nicht besser machen – höchstens noch schlimmer.
- Wir können also diesen MIN-Teilbaum "abschneiden", d.h. ignorieren
- Symmetrisch gilt: Wenn ein MAX-Knoten einen Zug machen kann, der v Punkte garantiert und $v \geq \beta$ ist, dann ist dieser Zug für den darüberliegenden MIN-Knoten schlechter als der vorher gefundene – und wird deshalb „gepruned"



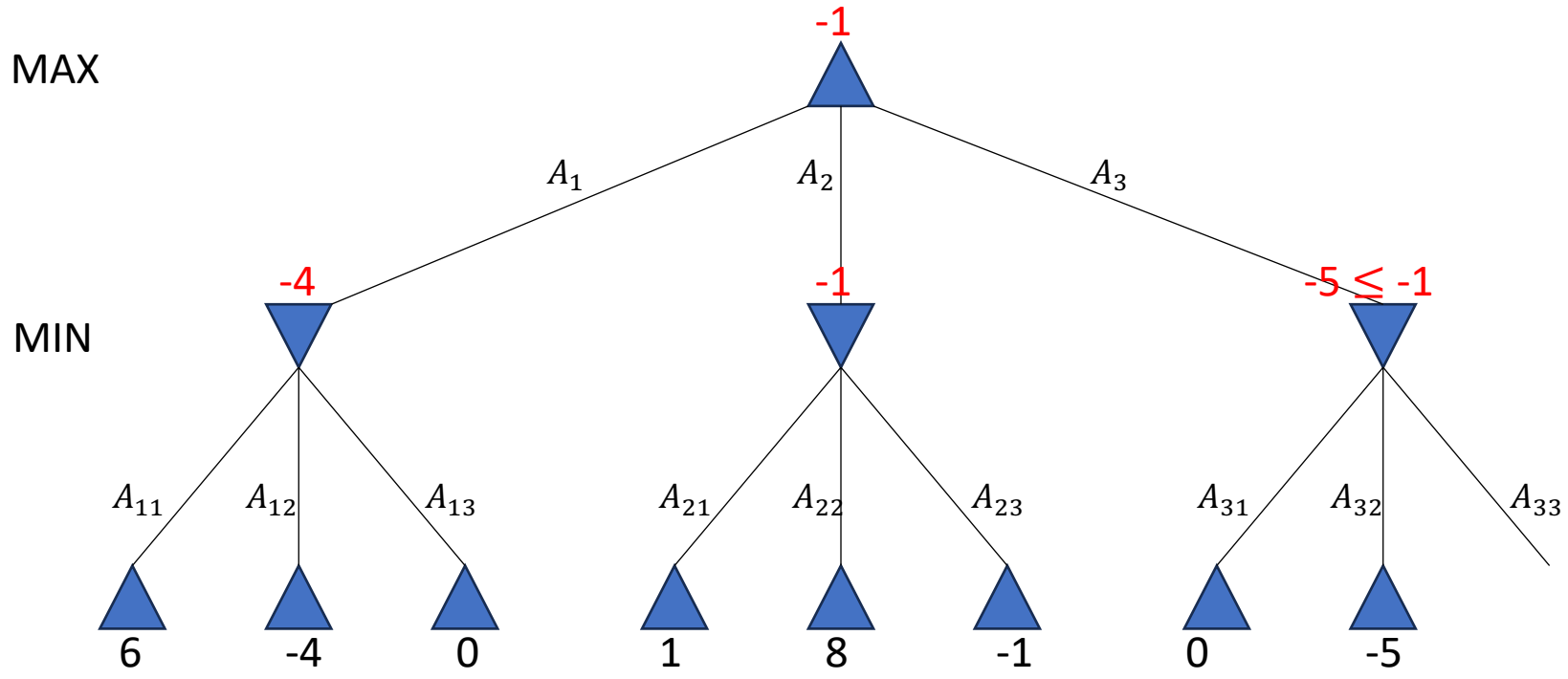


Wir betrachten jetzt die Aufgaben von vorhin noch einmal.
An welchen Stellen könnte die Suche durch Alpha-Beta-Pruning vorzeitig beendet werden?

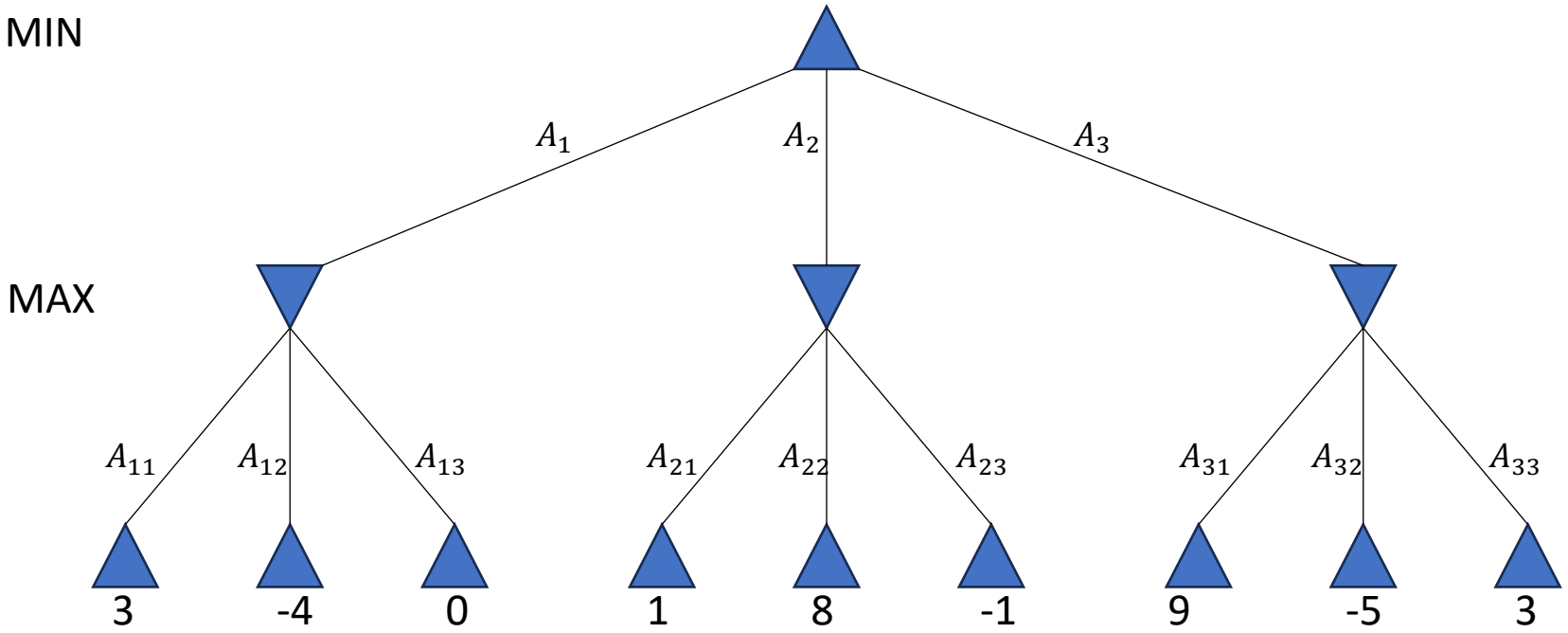
Aufgabe: Welchen Zug sollte MAX wählen und wie viele Punkte kann er erreichen?



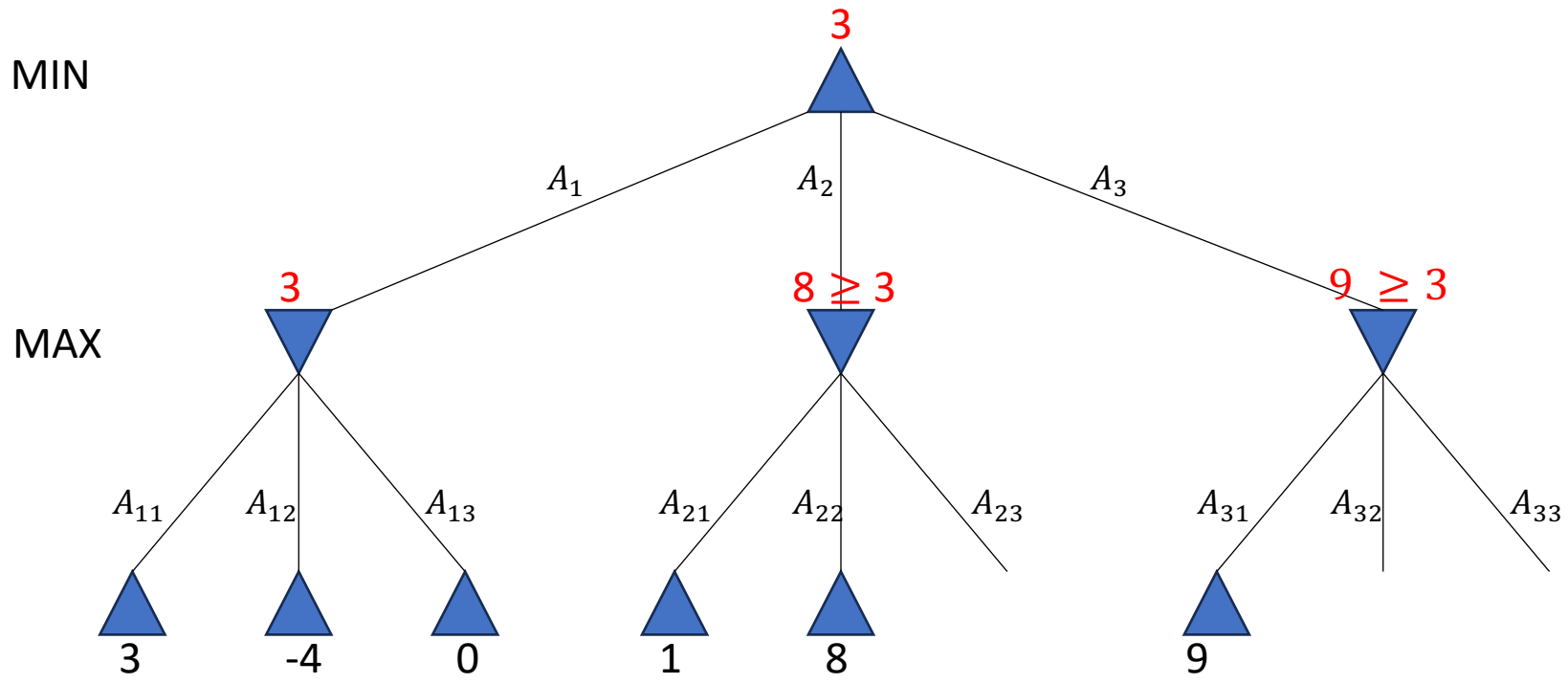
Lösung mit Alpha-Beta-Pruning



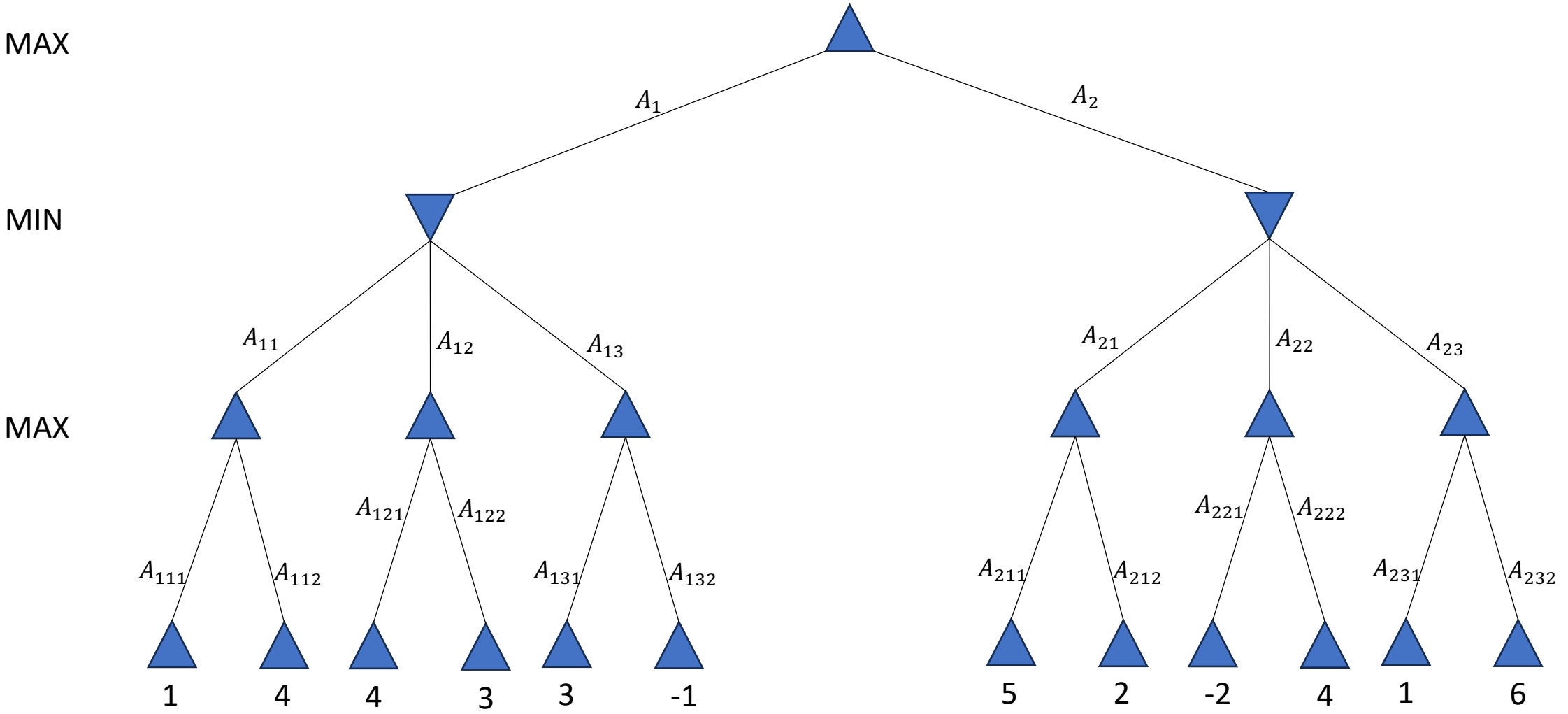
Aufgabe: Welchen Zug sollte MIN wählen und wie viele Punkte kann er erreichen?



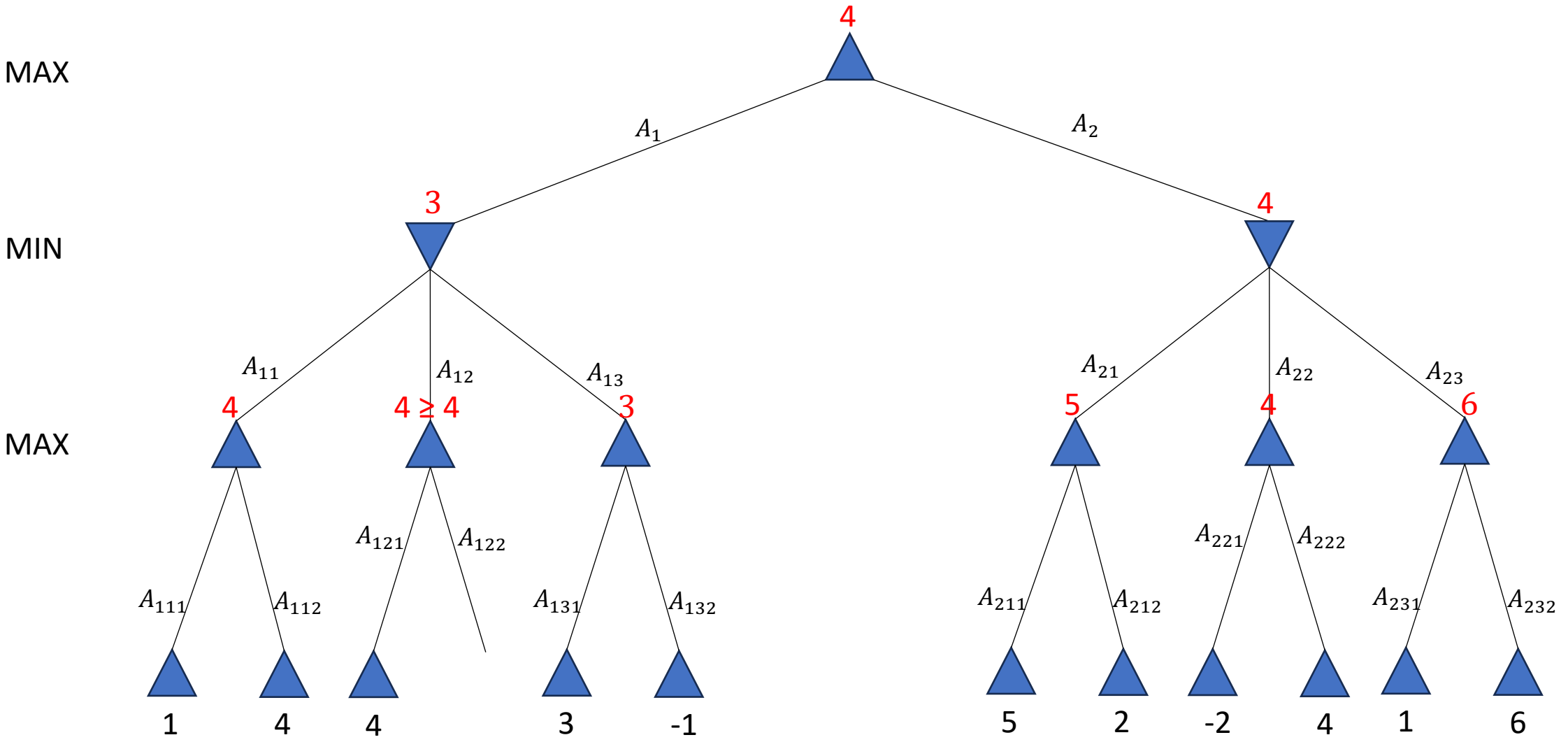
Lösung mit Alpha-Beta-Pruning



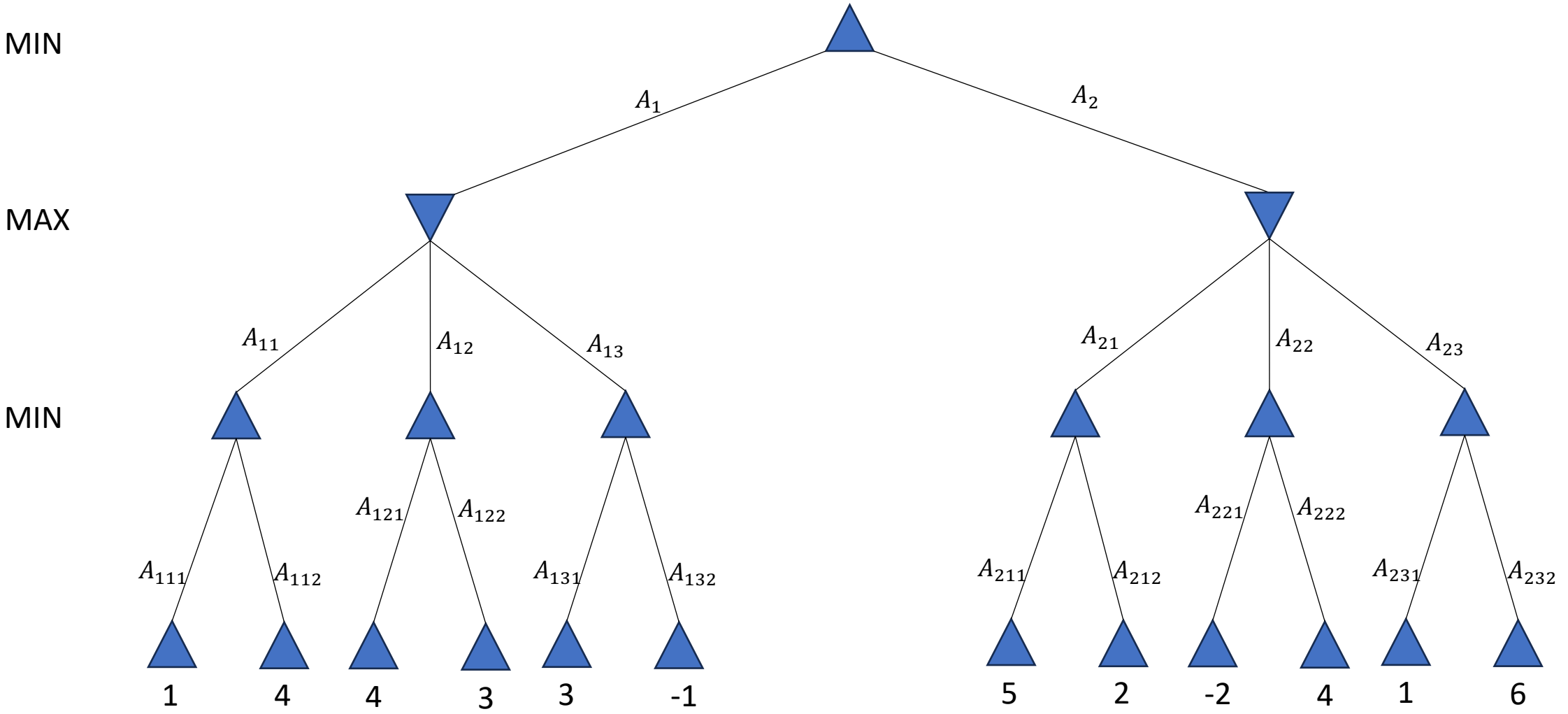
Aufgabe: Welchen Zug sollte MAX wählen und wie viele Punkte kann er erreichen?



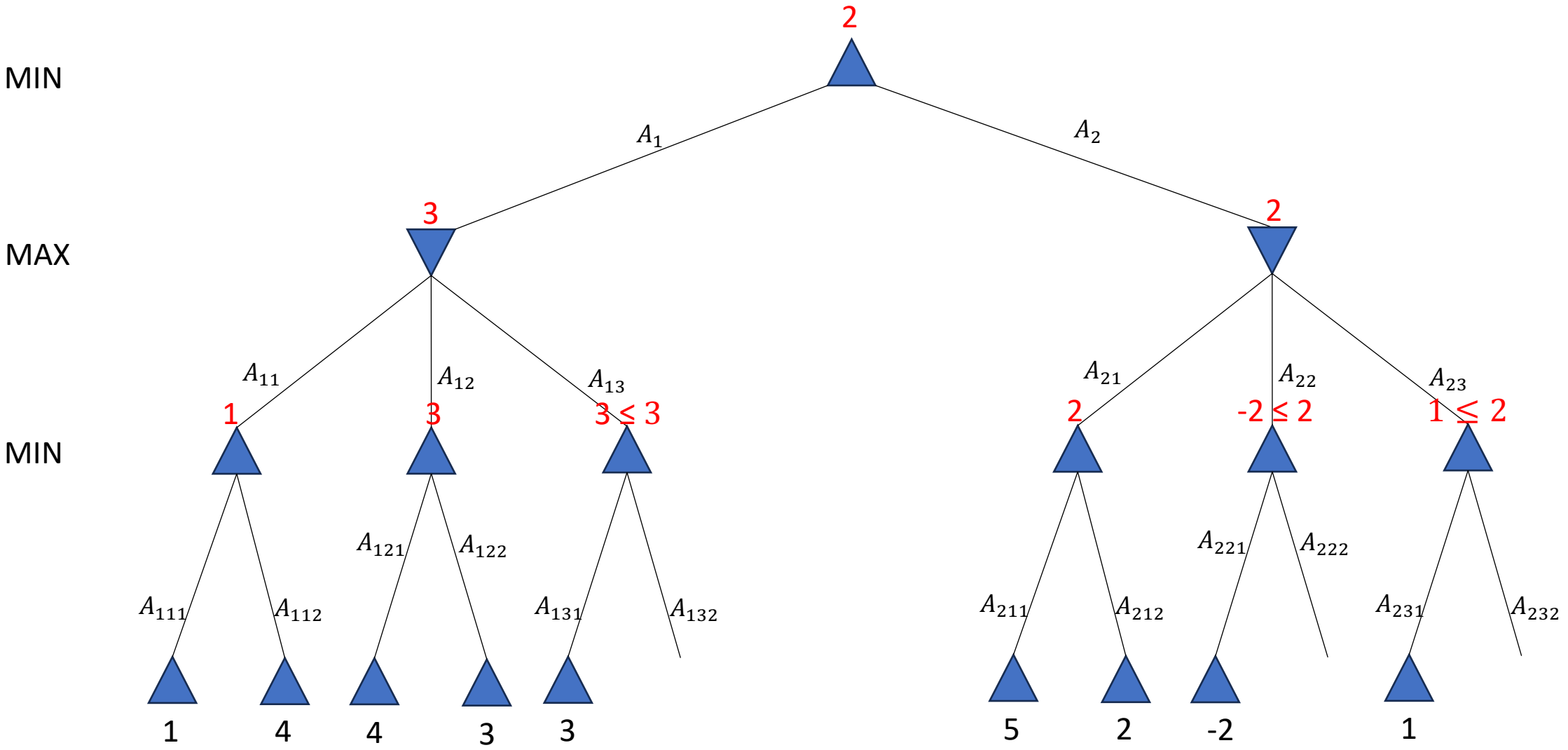
Lösung mit Alpha-Beta-Pruning



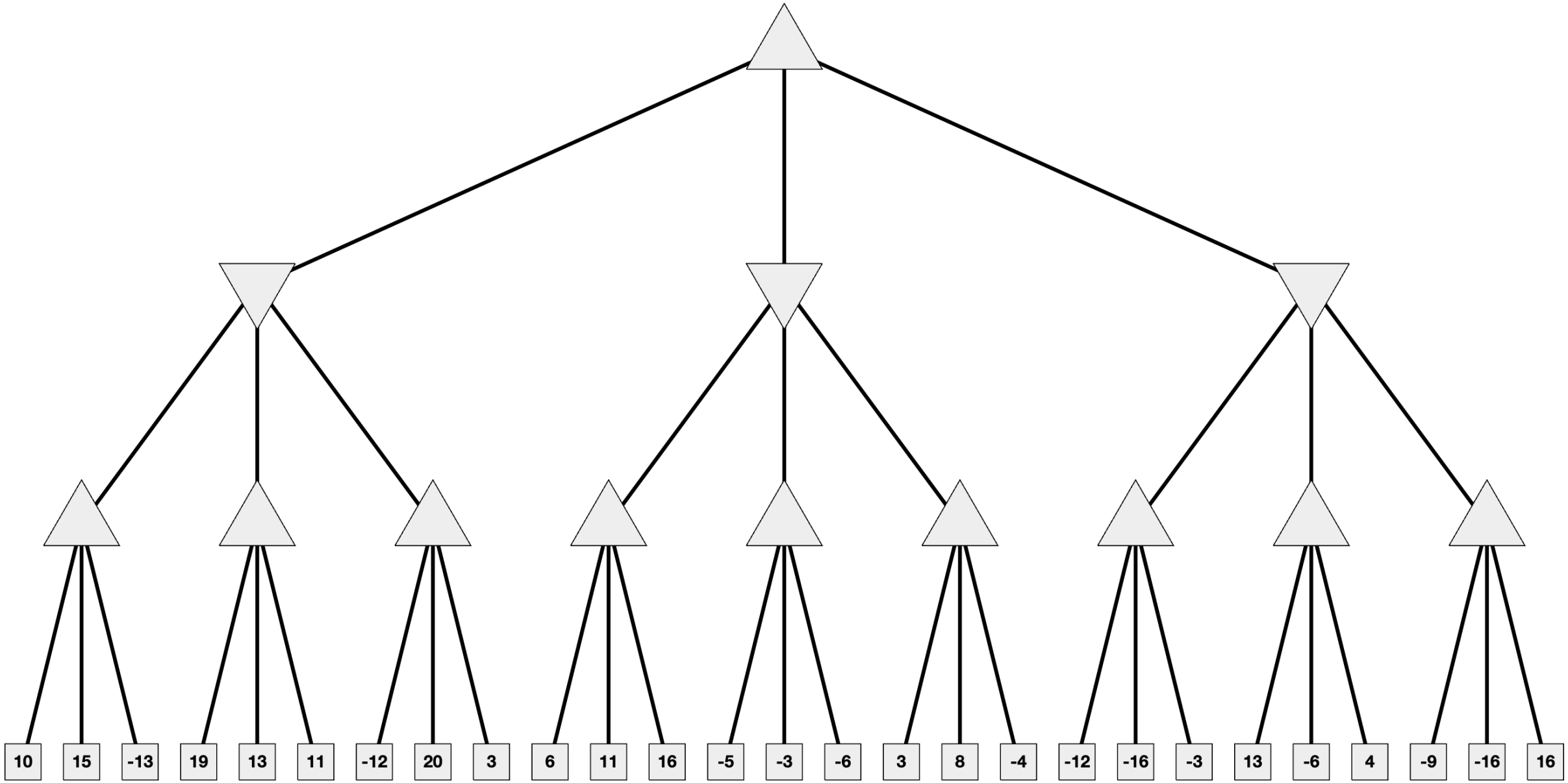
Aufgabe: Welchen Zug sollte MIN wählen und wie viele Punkte kann er erreichen?



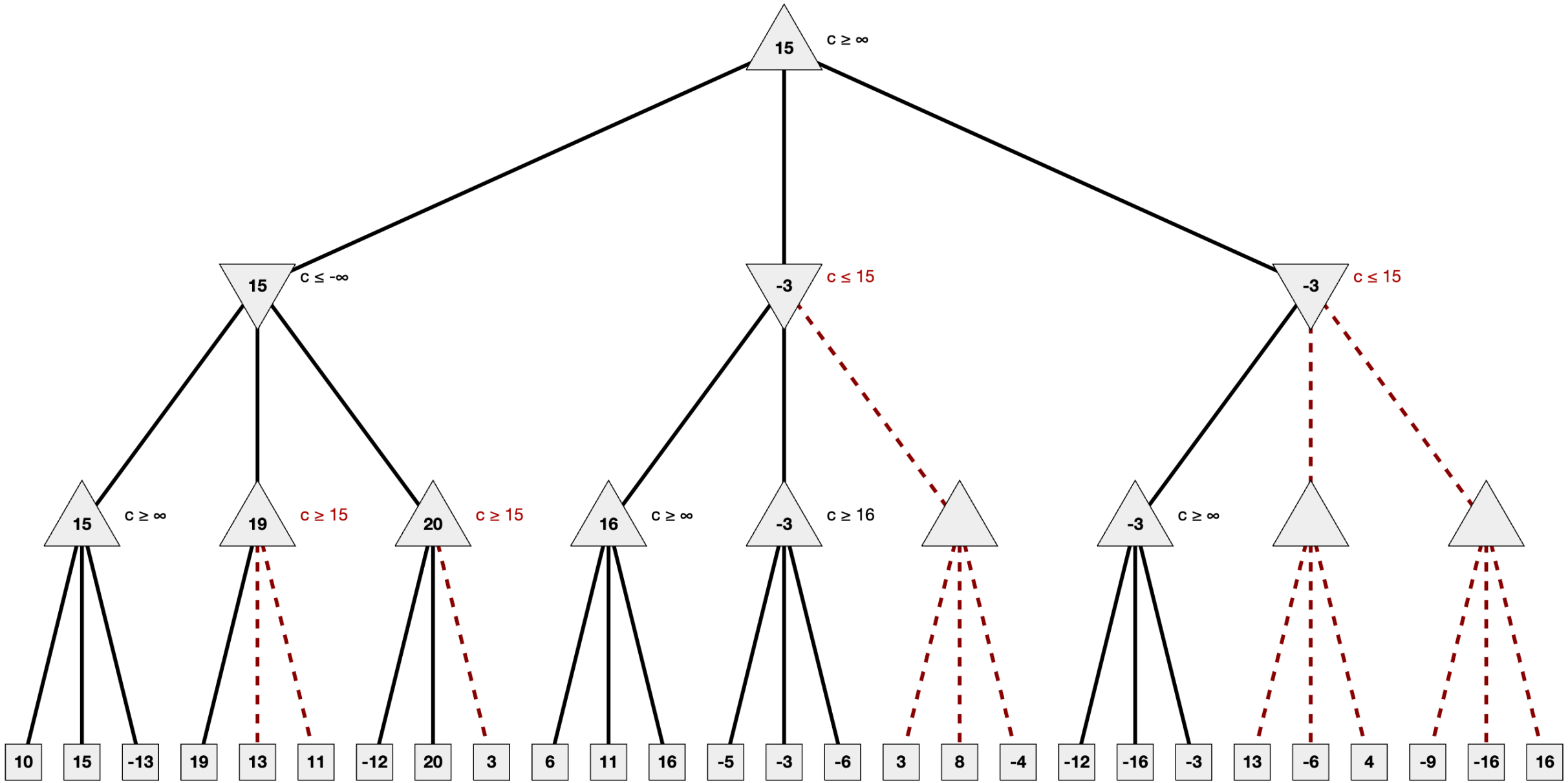
Lösung mit Alpha-Beta-Pruning

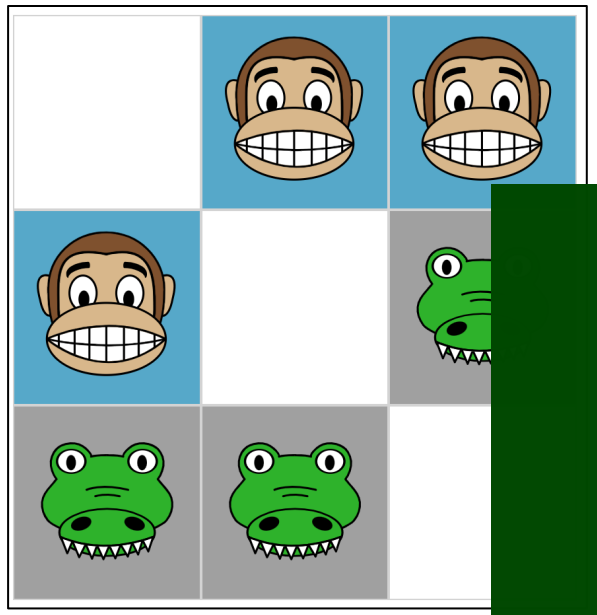


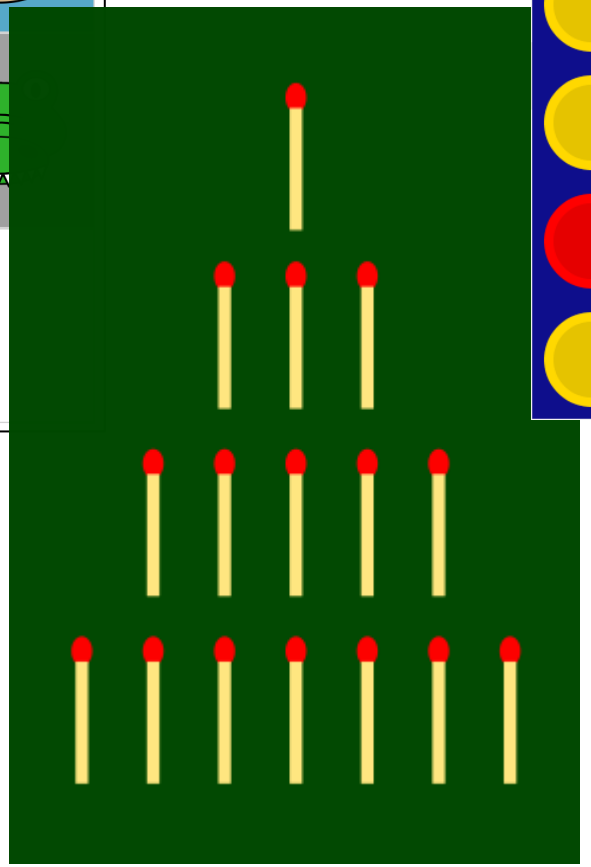
Löse mit Alpha-Beta-Pruning

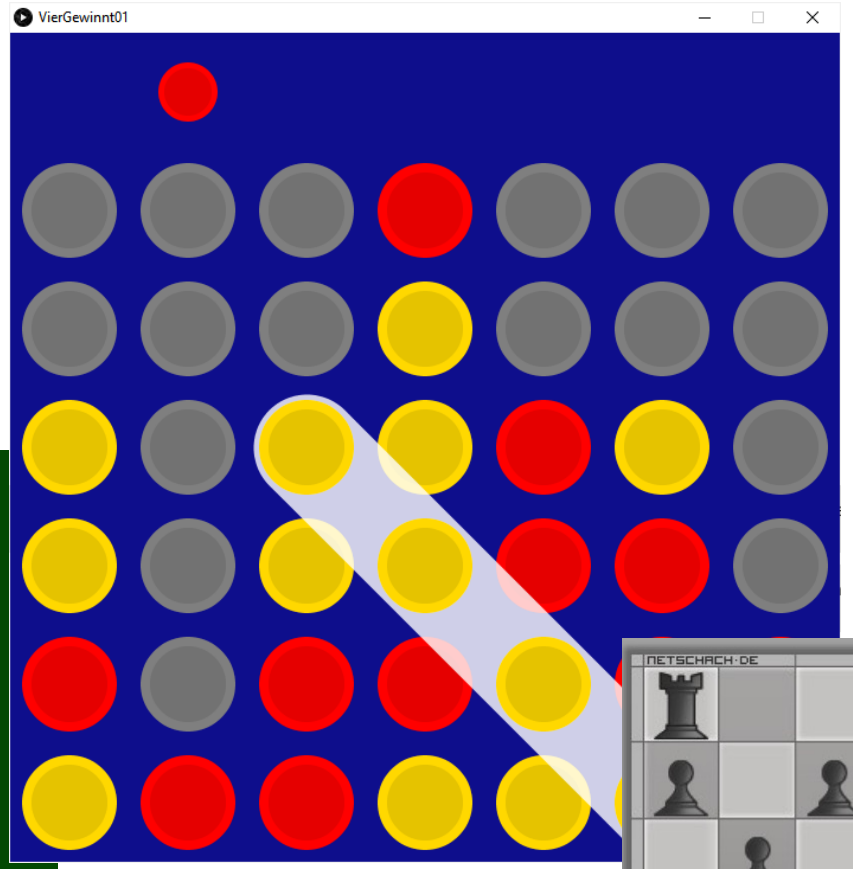


Lösung


15
c ≥ ∞
15
c ≤ -∞
-3
c ≤ 15
-3
c ≤ 15
15
c ≥ ∞
19
c ≥ 15
20
c ≥ 15
16
c ≥ ∞
-3
c ≥ 16
-3
c ≥ ∞
10
15
-13
19
13
11
-12
20
3
6
11
16
-5
-3
-6
3
8
-4
-12
-16
-3
13
-6
4
-9
-16
16






VierGewinnt01


NETSCHACH·DE
8
7
6
5
4
3
2
1
A
B
C
D
E
F
G
H