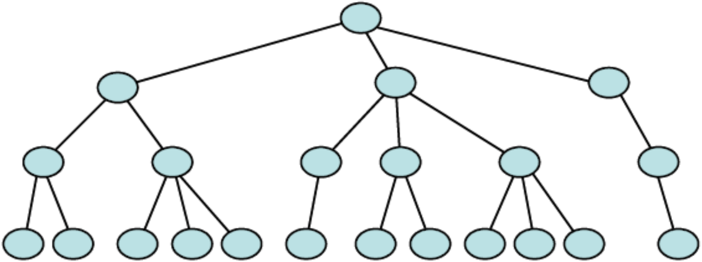
Informatik

Spielbaum

In einem **Mehrwegebaum** kann jeder Knoten eine variable Anzahl von Söhnen besitzen.



Ein **Spielbaum** ist ein Mehrwegebaum mit zwei Typen von Knoten: Minimum-Knoten und Maximum-Knoten.

Die Knoten repräsentieren Spielstellungen.

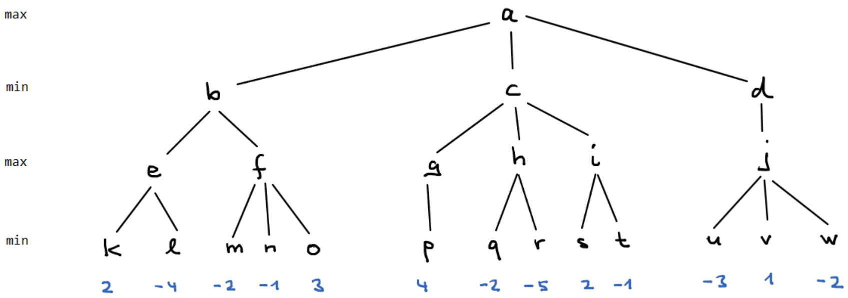
Die Kanten zwischen den Knoten repräsentieren Spielzüge.

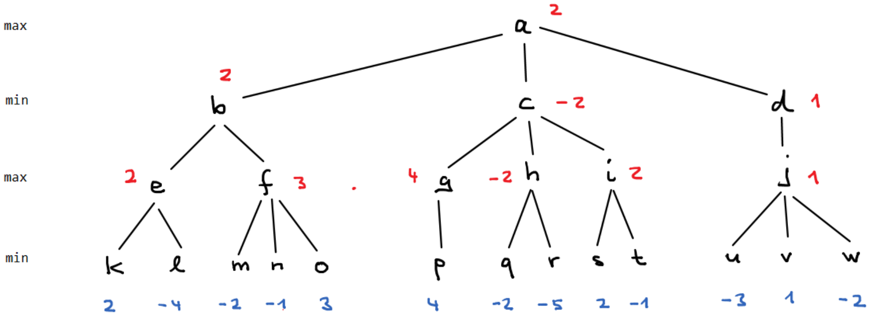
Der Spielbaum eignet sich für Zwei-Personen Nullsummenspiele wie Schach, Dame, TicTacToe, wo 2 Personen abwechselnd ziehen und der Gewinn des einen Spielers dem Verlust des anderen Spielers entspricht.

Der Wert eines Minimum-Knotens ist das Minimum der Werte seiner Söhne. Der Wert eines Maximum-Knotens ist das Maximum der Werte seiner Söhne.

Der Wert eines Blattes wird bestimmt durch eine statische Stellungsbewertung.

Beispiel für einen Spielbaum





Reihenfolge, in der die Knoten ihre Bewertung erhalten:

e:2 f:3 b:2 g:4 h:-2 i:2 c:-2 j:1 d:1 a:2
bester zug: b

Der MinMax-Algorithmus

```
def maximize(state):
    '''
    state: Spielstellung
    returns: (st, k), die Folgestellung st, die die höchste Bewertung k hat
    '''
    Falls state ein Blatt ist, wird keine Folgestellung sondern nur die
        Stellungsbewertung zurückgegeben.

    Von allen Kindern von state wird das mit der höchsten
    Bewertung des Minimizers zurückgegeben.


def minimize(state):
    '''
    state: Spielstellung
    returns: (st, k), die Folgestellung st, die die niedrigste Bewertung k hat
    '''
    Falls state ein Blatt ist, wird keine Folgestellung sondern nur die
        Stellungsbewertung zurückgegeben.

    Von allen Kindern von state wird das mit der niedrigsten
    Bewertung des Maximizers zurückgegeben.

st = ...                           # Ausgangsstellung
besterZug, k = maximize(st)        # der beste Zug nach st für den Maximizer
```

Für Implementierung des MinMax-Algorithmus setzen wir folgende Funktionen voraus:

```
def nextstates(state):
    '''
    state: Spielstellung
    returns: Liste mit möglichen Folgestellungen
    '''

def terminal_test(state):
    '''
    state: Spielstellung
    returns: True, wenn Stellung ein Blatt ist
    '''

def evaluation(state):
    '''
    state: Spielstellung
    returns: Zahl, die den Wert der Stellung wiedergibt
    '''
```

Mit diesen Funktionen können wir die Funktionen maximize und minimize implementieren unabhängig von der konkreten Instanz des MinMax-Problems.

```python
inf = float('inf')
def maximize(state):
    if terminal_test(state):
        return None, evaluation(state)
    v, bestChild = -inf, None
    for child in nextstates(state):
        _, utility = minimize(child)
        if utility > v:
            bestChild, v = child, utility
    return bestChild, v

def minimize(state):
    if terminal_test(state):
        return None, evaluation(state)
    v, bestChild = inf, None
    for child in nextstates(state):
        _, utility = maximize(child)
        if utility < v:
            bestChild, v = child, utility
    return bestChild, v

# bester folgezug für maximizer nach state
bestnext, _ = maximize(state)

# bester folgezug für minimizer nach state
bestnext, _ = minimize(state)
```

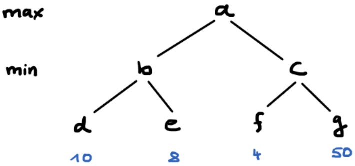
Beispiel-Spielbäume geben wir mit zwei dictionaries an:

```
nxt = {'a':list('bc'),'b':list('de'),'c':list('fg')}   # wurzel 'a'
blatt = {'d':10,'e':8,'f':4,'g':50}

def nextstates(state):
    return nxt[state]

def terminal_test(state):
    return state in blatt

def evaluation(state):
    return blatt[state]
```



Reihenfolge der besuchten Knoten (ohne Blätter): b:8 c:4 a:8
Bester Zug: b

Für ein konkretes Spiel müssen wir uns entscheiden, wie wir eine Spielstellung repräsentieren. Anschließend müssen wir die drei Funktionen `nextstates`, `terminal_test` und `evaluation` implementieren.

Beispiel: Tic-Tac-Toe

Es gibt mehrere Möglichkeiten, eine Spielstellung zu repräsentieren.

```
x . .
. o .
. . .
```

- Matrix mit Zeichen: `[['x','.','.'],['.','o','.'],['.','.','.']]`
- Matrix mit Zahlen: `[[1,0,0],[0,-1,0],[0,0,0]]`
- Liste mit Tupeln: `[(0,0),(1,1)]`
- Liste mit Zahlen: `[0,4]`

usw.

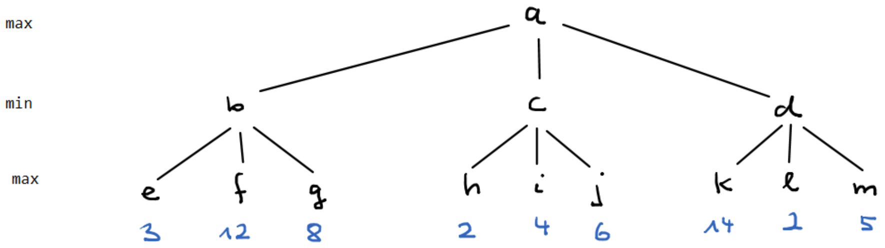
Beispiel: Liste mit Zahlen, d.h. die Felder des Spielfeldes werden von 0 bis 8 durchnummeriert.

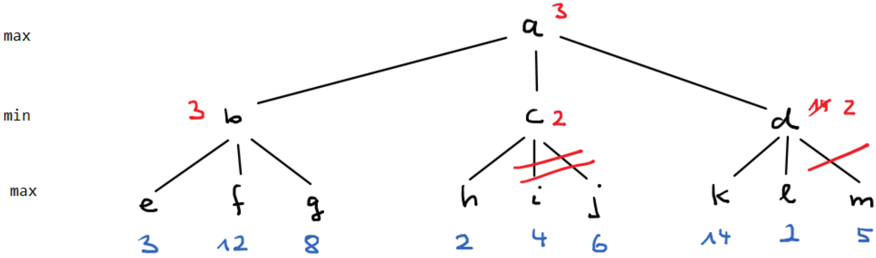
```python
def nextstates(state):
    temp = []
    for i in range(9):
        if i not in state:
            temp.append(state+[i])
    return temp

def evaluation(state):
    '''
    state: Stellung
    returns: Zahl, die den Wert der Stellung wiedergibt
      +1 Maximizer gewinnt, -1 Minimizer gewinnt, 0 Sonst
    '''
    # your code here
    pass

def terminal_test(state):
    w = evaluation(state)
    return w == 1 or w == -1 or len(state) == 9
```

Der Alpha-Beta Algorithmus versucht Teile des Baumes abzuschneiden (pruning), die erkennbar nicht mehr durchsucht werden müssen.





Dazu wird jedem Knoten ein α und ein β-Wert mitgegeben, die im Verlauf des Algorithmus upgedated werden und die es gestatten, ein Kriterium zu formulieren, wann das pruning erfolgen kann.

Der Alpha-Beta-Algorithmus

```
inf = float('inf')
def maximize(state,alpha,beta):
    if terminal_test(state):
        return None,evaluation(state)
    v, bestChild =-inf, None
    for child in nextstates(state):
        _,utility = minimize(child,alpha,beta)
        if utility > v:
            v, bestChild = utility, child
        if v >= beta:
            break
        if v > alpha:
            alpha = v
    return bestChild, v

def minimize(state,alpha,beta):
    if terminal_test(state):
        return None,evaluation(state)
    v, bestChild = inf, None
    for child in nextstates(state):
        _,utility = maximize(child,alpha,beta)
        if utility < v:
            v, bestChild = utility, child
        if v <= alpha:
            break
        if v < beta:
            beta = v
    return bestChild, v

# bester folgezug für maximizer nach state
bestnext, _ = maximize(state,-inf,inf)
```

Vereinfachte Notation zur Durchführung des Alpha-Beta-Algorithmus:

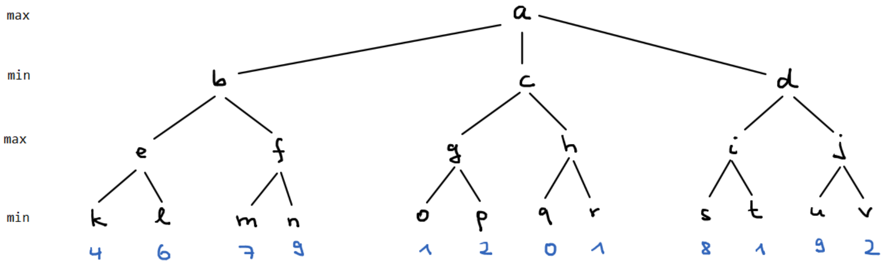
Zu Beginn ist $\alpha = -\infty$ und $\beta = +\infty$.

α und β werden von oben nach unten durchgereicht. Bei den max-Knoten wird β nicht verändert, bei den min-Knoten wird α nicht verändert. Wir notieren deshalb nur die α-Werte bei den max-Knoten und nur die β-Werte bei den min-Knoten. An den Blätter notieren wir keine α oder β-Werte.

Der Algorithmus ist eine Tiefensuche, die abwärts- und aufwärts-Bewegungen macht. Abwärts erben die Kinder ihre α und β-Werte von den Großeltern. Wenn aufwärts ein Eltern-Knoten bei einem Kind etwas sieht, was er haben will, nimmt er es von dem Kind.

Wenn an einer Stelle entdeckt wird, dass α größer-gleich dem darüber stehenden β ist oder dass β kleiner-gleich dem darüber stehenden α ist, wird gepruned.

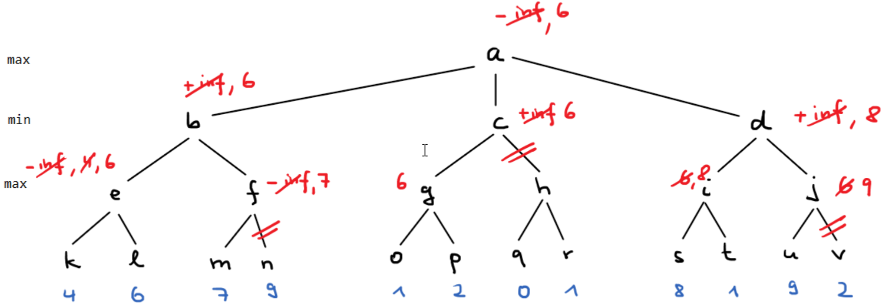
Der Auftraggeber des pruning ist der Vater des Knotens, bei dem das pruning erfolgt.



$\downarrow$ α, β von Großeltern

$\uparrow$ besserer Wert von Kind

$\alpha \geq \beta$ oder $\beta \leq \alpha$: pruning



Reihenfolge des Blattbesuchs, # für pruning:

k l m # o p # s t u #
bester Zug d