

# 2.5 Bäume

- 2.5.1 Binäre Suchbäume
- 2.5.2 Optimale Suchbäume
- 2.5.3 Balancierte Bäume
- 2.5.4 Skip-Listen
- 2.5.5 Union-Find-Strukturen

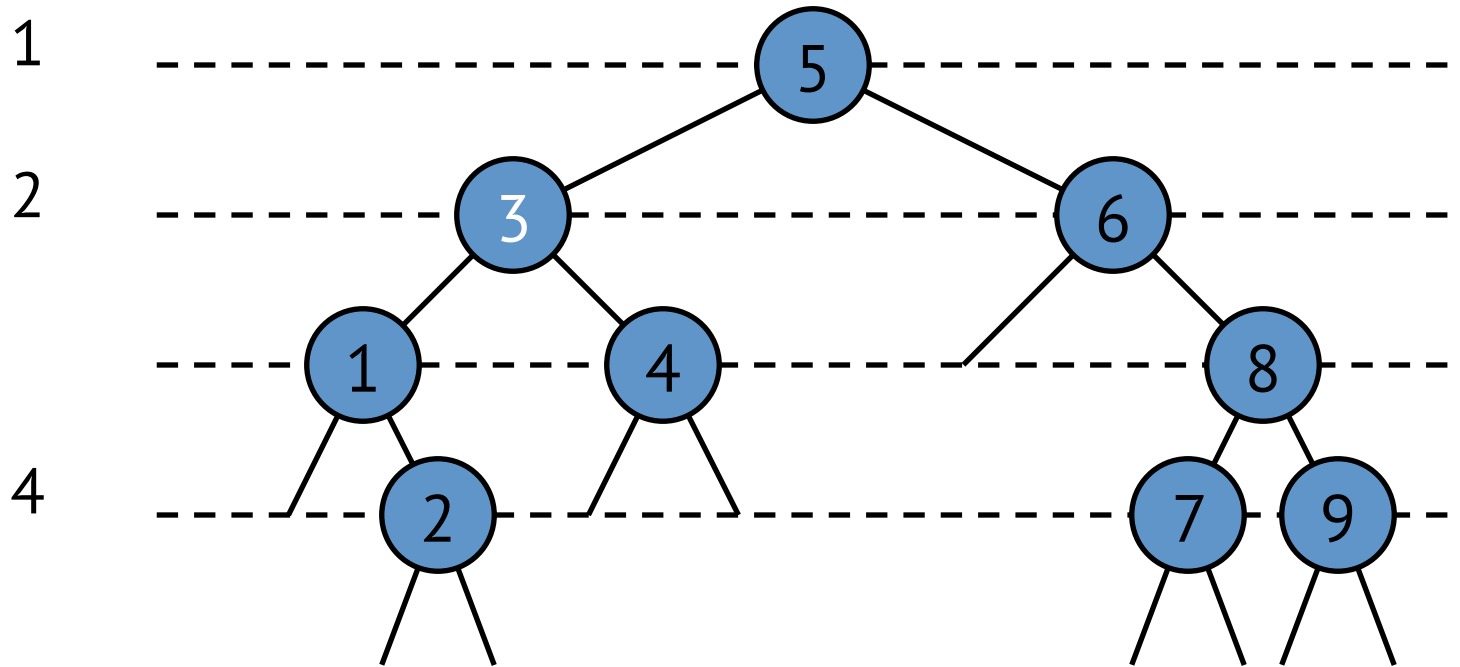


# Optimale Suchbäume

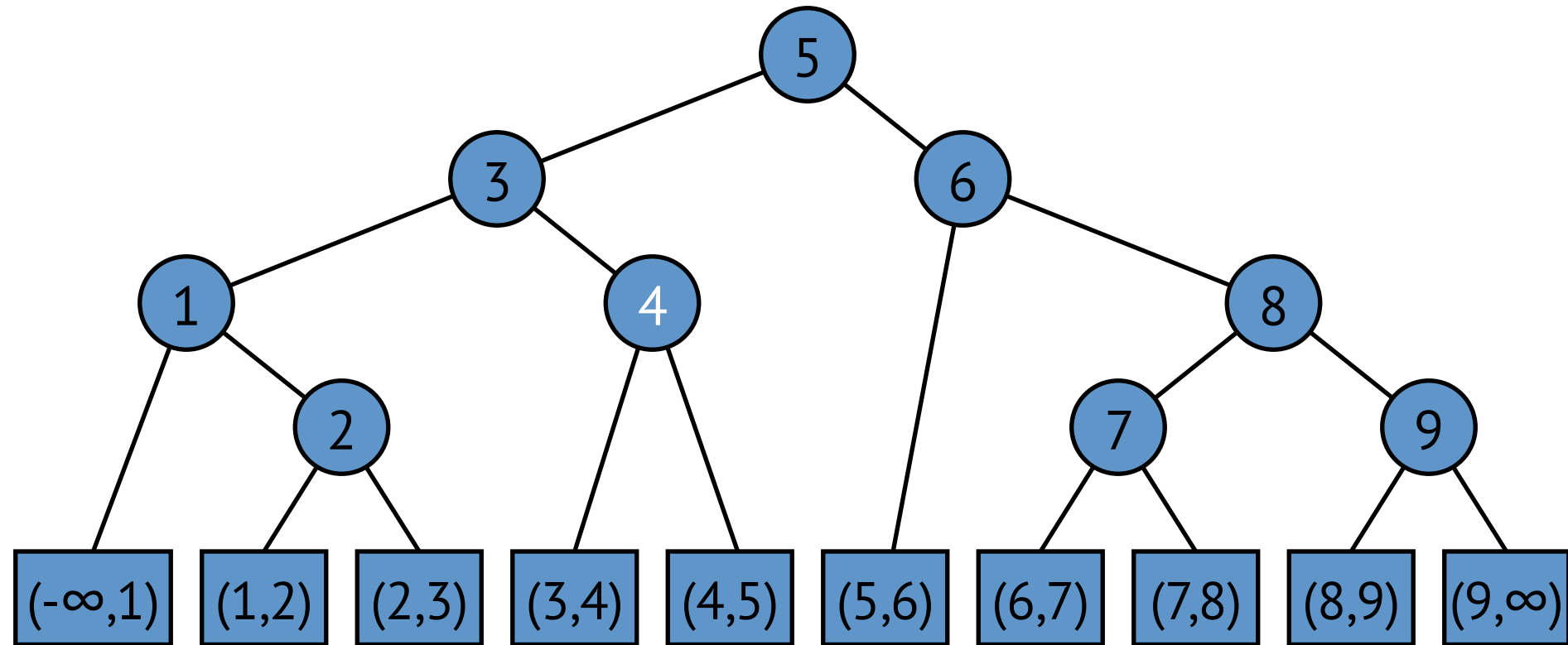
- In manchen Anwendungen ist die Zugriffswahrscheinlichkeit auf die Elemente a priori bekannt.
- Versuche die Elemente, auf die am häufigsten zugegriffen wird, in der Nähe der Wurzel zu speichern.
- Beispiel: Worthäufigkeiten, Übersetzungstabelle



# Optimale Suchbäume



# Optimale Suchbäume



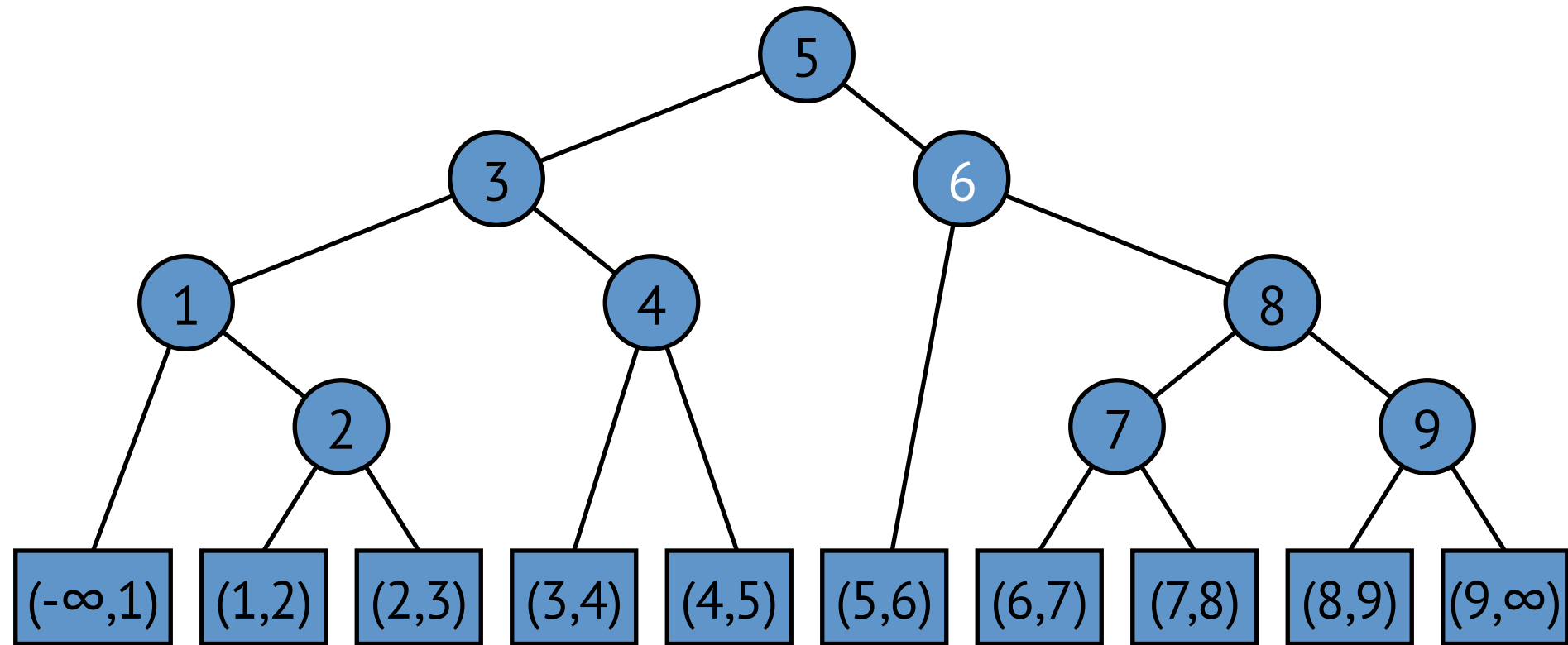
Explizite Blätter für negative Suchanfragen

# Optimale Suchbäume

- Seien  $N_1 \dots N_n$  die inneren Knoten des Suchbaums
- Dann gibt es genau  $n+1$  Blätter  $l_0 \dots l_n$
- Es sei  $p_i$  die Wahrscheinlichkeit, dass auf Knoten  $N_i$  zugegriffen wird
- Es sei  $q_i$  die Wahrscheinlichkeit, dass auf Knoten  $l_i$  zugegriffen wird, d.h. nach einem Schlüssel  $X$  mit  $N_i.X < X < N_{i+1}.X$  gesucht wird



# Optimale Suchbäume



# Aufwand

$$\sum_{i=1}^n p_i + \sum_{i=0}^n q_i = 1$$

$$\begin{aligned} C_{avg} &= \sum_{i=1}^n (\text{depth}(N_i) + 1)p_i + \sum_{i=0}^n (\text{depth}(I_i) + 1)q_i \\ &= 1 + \sum_{i=1}^n \text{depth}(N_i)p_i + \sum_{i=0}^n \text{depth}(I_i)q_i \end{aligned}$$

# Aufwand

- Betrachte einen Teilbaum mit Wurzel  $N_k$  des optimalen Suchbaums
- Dieser Baum enthält eine Teilmenge  $N_i \dots N_j$  der Knoten und  $I_{i-1} \dots I_j$  der Blatt-Intervalle
- Dieser Teilbaum ist ebenfalls ein optimaler Suchbaum (cut & paste)





- Die Wahrscheinlichkeit, dass auf den Teilbaum mit Wurzel  $N_k$  zugegriffen wird ist

$$w(i, j) = \sum_{l=i}^j p_l + \sum_{l=i-1}^j q_l$$

- Seien  $C(i, j)$  die Kosten, die durch den Teilbaum  $N_i \dots N_j$  verursacht werden



# Aufwand

- Der Teilbaum mit Wurzel  $N_k$  besitzt zwei weitere Teilbäume mit den Knoten
  - $N_i \dots N_{k-1}$  und  $l_{i-1} \dots l_{k-1}$
  - $N_{k+1} \dots N_j$  und  $l_k \dots l_j$die jeder für sich wieder optimal sind
- Jeder Pfad im Teilbaum  $N_i \dots N_{k-1}$  wird durch  $N_k$  um einen Schritt verlängert

# Rekursive Optimierung

$$\begin{aligned} C(i, j) &= p_k \\ &+ C(i, k - 1) + w(i, k - 1) \\ &+ C(k + 1, j) + w(k + 1, j) \end{aligned}$$

$$w(i, j) = w(i, k - 1) + p_k + w(k + 1, j)$$

$$C(i, j) = C(i, k - 1) + C(k + 1, j) + w(i, j)$$

$$C(i, i - 1) = q_{i-1}$$

- Für den Teilbaum  $N_i \dots N_j$  soll der optimale Wurzelknoten bestimmt werden...

$$C(i, i - 1) = q_{i-1}$$

$$C(i, j) = \min_{i \leq k \leq j} \{C(i, k - 1) + C(k + 1, j)\} + w(i, j)$$

- Dynamische Programmierung!

- Speichere Zwischenergebnisse in Tabellen
  - $C[i,j]$  : Kosten des optimalen Suchbaums für die Knoten  $N_i \dots N_j$  und  $l_{i-1} \dots l_j$
  - $W[i,j]$  : Zugriffswahrscheinlichkeit für den entsprechenden Teilbaum
  - $R[i,j]$  : Wurzelknoten des optimalen Suchbaums

# Aufwand

OptTree(P[1..n], Q[0..n])

for i ← 1 to n+1 do

    C[i, i-1] ← Q[i-1]

    W[i, i-1] ← Q[i-1]

for l ← 1 to n do

    for i ← 1 to n-l+1 do

        j ← i+l-1

        W[i, j] ← W[i, j-1] + P[j] + Q[j]

        C[i, j] ← min{ C[i, k-1] + C[k+1, j] } + W[i, j]

        R[i, j] ← argmin{ C[i, k-1] + C[k+1, j] }

# Aufwand

- Der optimale Suchbaum muss nicht immer minimale Höhe besitzen.
- Das häufigst gefragte Element muss nicht der Wurzelknoten sein.
- Berechnungsaufwand mit DP:  $O(n^3)$
- Lohnt nur für viele Anfragen
- Wahrscheinlichkeiten können aus der Zugriffsstatistik extrapoliert werden.

