Die Höhe eines Suchbaums ist für seine Geschwindigkeit maßgebend.

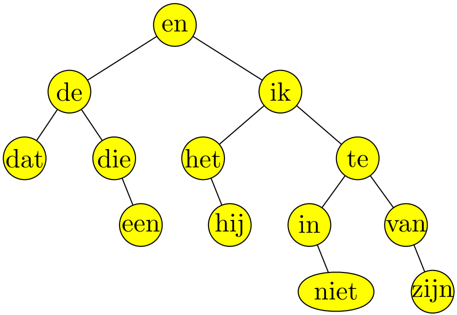
Theorem

Die Operationen Einfügen, Löschen und Suchen benötigen $O(h)$ Zeit bei einem binären Suchbaum der Höhe h.

RWTH AACHEN UNIVERSITY

Optimale Suchbäume

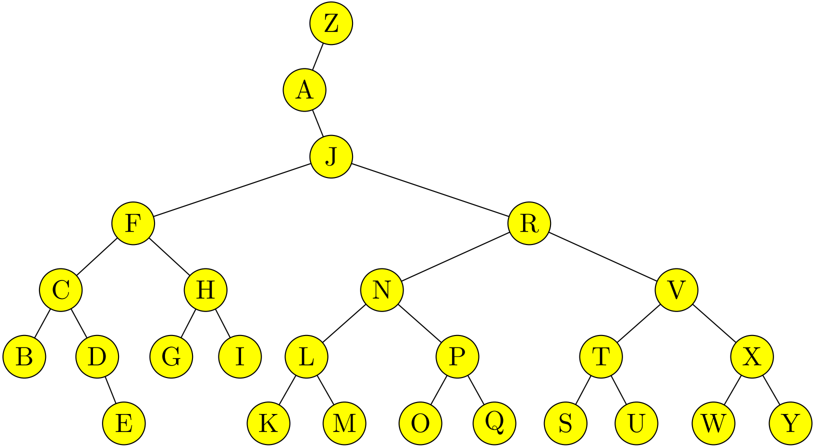
Ein optimaler Suchbaum für die 13 häufigsten Wörter des Buches Max Havelaar.



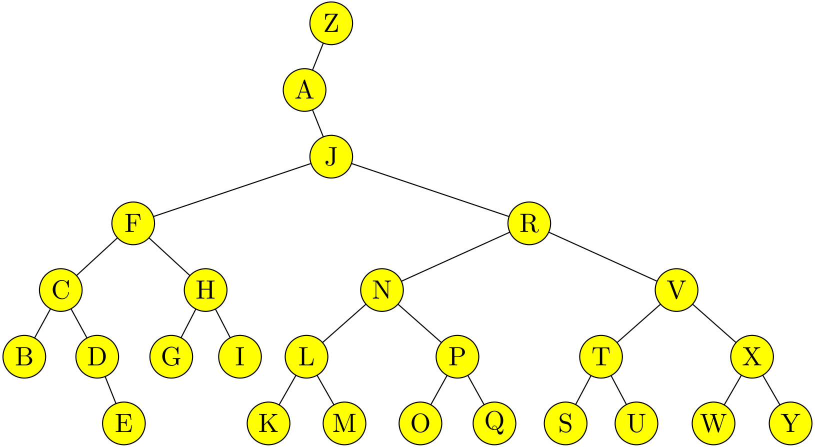
Wort	de	en	het	van	ik	te	dat	die
Anzahl	4770	2709	2469	2259	1999	1935	1875	1807

(Häufigkeitstabelle nach Wikipedia)

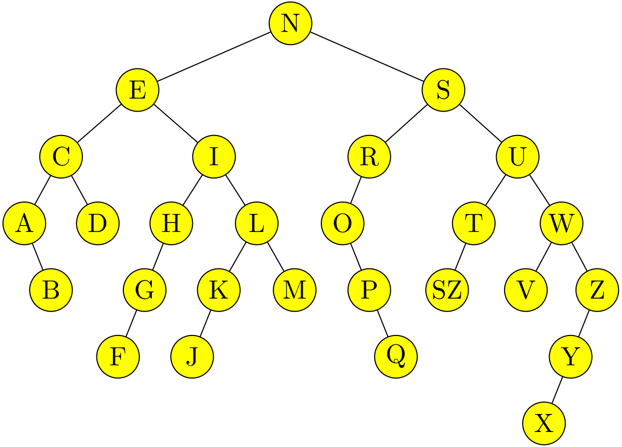
Ein optimaler Suchbaum enthält A bis Z. Die W'keit für A und Z sei 0.49, der Rest auf B–Y verteilt.
Wie sieht der Baum aus?



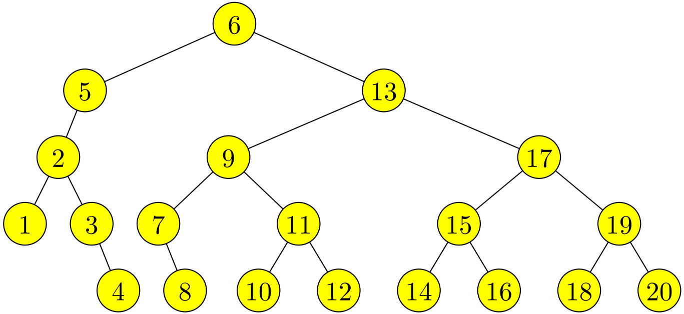
Ein optimaler Suchbaum enthält A bis Z. Die W'keit für A und Z sei 0.49, der Rest auf B–Y verteilt.
Wie sieht der Baum aus?



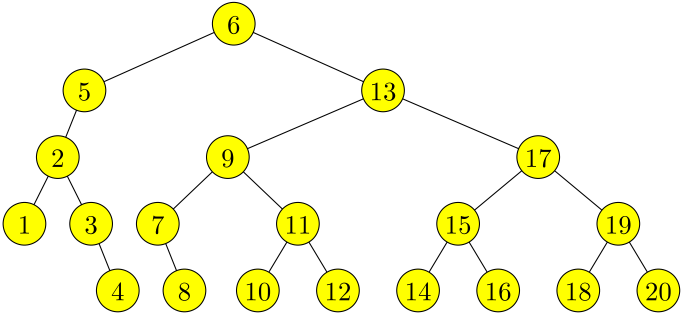
Optimale Suchbäume



Ein optimaler Suchbaum enthält die Zahlen 1–20. Auf jede wird mit W'keit 1/50 zugegriffen. Mit W'keit 3/5 wird 5.5 gesucht.
Wie sieht der Baum aus?



Ein optimaler Suchbaum enthält die Zahlen 1–20. Auf jede wird mit W'keit 1/50 zugegriffen. Mit W'keit 3/5 wird 5.5 gesucht.
Wie sieht der Baum aus?



Optimale Suchbäume

- Es seien $k_1 < \cdots < k_n$ Schlüssel.
- Wir suchen k_i mit Wahrscheinlichkeit p_i.

Es gelte $\displaystyle\sum_{i=1}^{n} p_i = 1$.

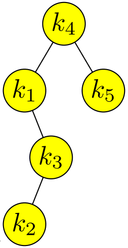
Definition

Ein optimaler Suchbaum für $k_1, \ldots, k_n$ gemäß den Zugriffswahrscheinlichkeiten $p_1, \ldots, p_n$ ist

- ein Suchbaum für $k_1, \ldots, k_n$,
- und hat eine minimale Anzahl von Vergleichen im Erwartungswert, falls k_i mit Wahrscheinlichkeit p_i gesucht wird.

Es sei $w_{i,j} = \sum_{k=i}^{j} p_k$.

Falls es in einem optimalen Suchbaum einen Unterbaum gibt, der die Schlüssel $k_i, \ldots, k_j$ enthält, dann sei $e_{i,j}$ der Erwartungswert der Anzahl der Vergleiche, die bei einer Suche in diesem Unterbaum durchgeführt werden.

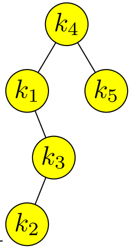


i	1	2	3	4	5
p_i	$1/4$	$1/8$	$1/6$	$1/4$	$5/24$

$w_{1,5} = 1$, $w_{2,3} = 7/24$, $e_{2,2} = 1/8$, $e_{2,3} = 10/24$, $e_{1,3} = 23/24$

Es sei $w_{i,j} = \sum_{k=i}^{j} p_k$.

Falls es in einem optimalen Suchbaum einen Unterbaum gibt, der die Schlüssel $k_i, \ldots, k_j$ enthält, dann sei $e_{i,j}$ der Erwartungswert der Anzahl der Vergleiche, die bei einer Suche in diesem Unterbaum durchgeführt werden.

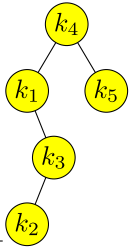


i	1	2	3	4	5
p_i	1/4	1/8	1/6	1/4	5/24

$w_{1,5} = 1$, $w_{2,3} = 7/24$, $e_{2,2} = 1/8$, $e_{2,3} = 10/24$, $e_{1,3} = 23/24$

Es sei $w_{i,j} = \sum_{k=i}^{j} p_k$.

Falls es in einem optimalen Suchbaum einen Unterbaum gibt, der die Schlüssel $k_i, \ldots, k_j$ enthält, dann sei $e_{i,j}$ der Erwartungswert der Anzahl der Vergleiche, die bei einer Suche in diesem Unterbaum durchgeführt werden.

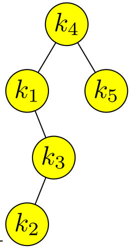


i	1	2	3	4	5
p_i	$1/4$	$1/8$	$1/6$	$1/4$	$5/24$

$w_{1,5} = 1$, $w_{2,3} = 7/24$, $e_{2,2} = 1/8$, $e_{2,3} = 10/24$, $e_{1,3} = 23/24$

Es sei $w_{i,j} = \sum_{k=i}^{j} p_k$.

Falls es in einem optimalen Suchbaum einen Unterbaum gibt, der die Schlüssel $k_i, \ldots, k_j$ enthält, dann sei $e_{i,j}$ der Erwartungswert der Anzahl der Vergleiche, die bei einer Suche in diesem Unterbaum durchgeführt werden.

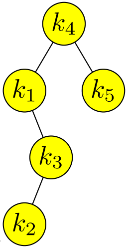


i	1	2	3	4	5
p_i	1/4	1/8	1/6	1/4	5/24

$w_{1,5} = 1$, $w_{2,3} = 7/24$, $e_{2,2} = 1/8$, $e_{2,3} = 10/24$, $e_{1,3} = 23/24$

Es sei $w_{i,j} = \sum_{k=i}^{j} p_k$.

Falls es in einem optimalen Suchbaum einen Unterbaum gibt, der die Schlüssel $k_i, \ldots, k_j$ enthält, dann sei $e_{i,j}$ der Erwartungswert der Anzahl der Vergleiche, die bei einer Suche in diesem Unterbaum durchgeführt werden.

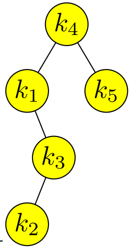


i	1	2	3	4	5
p_i	1/4	1/8	1/6	1/4	5/24

$w_{1,5} = 1$, $w_{2,3} = 7/24$, $e_{2,2} = 1/8$, $e_{2,3} = 10/24$, $e_{1,3} = 23/24$

Es sei $w_{i,j} = \sum_{k=i}^{j} p_k$.

Falls es in einem optimalen Suchbaum einen Unterbaum gibt, der die Schlüssel $k_i, \ldots, k_j$ enthält, dann sei $e_{i,j}$ der Erwartungswert der Anzahl der Vergleiche, die bei einer Suche in diesem Unterbaum durchgeführt werden.

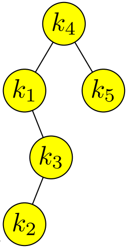


i	1	2	3	4	5
p_i	$1/4$	$1/8$	$1/6$	$1/4$	$5/24$

$w_{1,5} = 1$, $w_{2,3} = 7/24$, $e_{2,2} = 1/8$, $e_{2,3} = 10/24$, $e_{1,3} = 23/24$

Es sei $w_{i,j} = \sum_{k=i}^{j} p_k$.

Falls es in einem optimalen Suchbaum einen Unterbaum gibt, der die Schlüssel $k_i, \ldots, k_j$ enthält, dann sei $e_{i,j}$ der Erwartungswert der Anzahl der Vergleiche, die bei einer Suche in diesem Unterbaum durchgeführt werden.

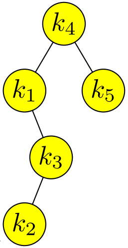


i	1	2	3	4	5
p_i	$1/4$	$1/8$	$1/6$	$1/4$	$5/24$

$w_{1,5} = 1,\ w_{2,3} = 7/24,\ e_{2,2} = 1/8,\ e_{2,3} = 10/24,\ e_{1,3} = 23/24$

Es sei $w_{i,j} = \sum_{k=i}^{j} p_k$.

Falls es in einem optimalen Suchbaum einen Unterbaum gibt, der die Schlüssel $k_i, \ldots, k_j$ enthält, dann sei $e_{i,j}$ der Erwartungswert der Anzahl der Vergleiche, die bei einer Suche in diesem Unterbaum durchgeführt werden.

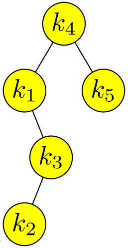


i	1	2	3	4	5
p_i	1/4	1/8	1/6	1/4	5/24

$w_{1,5} = 1$, $w_{2,3} = 7/24$, $e_{2,2} = 1/8$, $e_{2,3} = 10/24$, $e_{1,3} = 23/24$

Es sei $w_{i,j} = \sum_{k=i}^{j} p_k$.

Falls es in einem optimalen Suchbaum einen Unterbaum gibt, der die Schlüssel $k_i, \ldots, k_j$ enthält, dann sei $e_{i,j}$ der Erwartungswert der Anzahl der Vergleiche, die bei einer Suche in diesem Unterbaum durchgeführt werden.

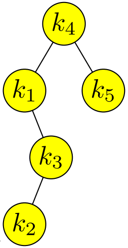


i	1	2	3	4	5
p_i	$1/4$	$1/8$	$1/6$	$1/4$	$5/24$

$w_{1,5} = 1$, $w_{2,3} = 7/24$, $e_{2,2} = 1/8$, $e_{2,3} = 10/24$, $e_{1,3} = 23/24$

Es sei $w_{i,j} = \sum_{k=i}^{j} p_k$.

Falls es in einem optimalen Suchbaum einen Unterbaum gibt, der die Schlüssel $k_i, \ldots, k_j$ enthält, dann sei $e_{i,j}$ der Erwartungswert der Anzahl der Vergleiche, die bei einer Suche in diesem Unterbaum durchgeführt werden.



i	1	2	3	4	5
p_i	$1/4$	$1/8$	$1/6$	$1/4$	$5/24$

$w_{1,5} = 1$, $w_{2,3} = 7/24$, $e_{2,2} = 1/8$, $e_{2,3} = 10/24$, $e_{1,3} = 23/24$

Wie können wir $w_{i,j} = \sum_{k=i}^{j} p_k$ berechnen?

Algorithmus

```
for i = 1, ..., n do
  for j = 1, ..., n do
    w[i, j] := 0;
    for k = i, ..., j do
      w[i, j] := w[i, j] + p[k]
    od
  od
od
```

Wie schnell ist dieser Algorithmus?

Geht es schneller?

Wie können wir $w_{i,j} = \sum_{k=i}^{j} p_k$ schneller berechnen?

Durch dynamische Programmieren:

Java

```
for i = 1, ..., n do
  w[i, i] := p[i];
  for j = i + 1, ..., n do
    w[i, j] := w[i, j − 1] + p[j]
  od
od
```

Wie schnell ist dieser Algorithmus?

Warum ist er korrekt?

Wie berechnen wir $e_{i,j}$?

Lemma

- $e_{i,j} = 0$ *für* $i > j$
- $e_{i,j} = \min_{i \le r \le j} (e_{i,r-1} + e_{r+1,j}) + w_{i,j}$ *für* $i \le j$

Beweis.

Ein Baum mit den Knoten $k_i, \dots, k_j$ hat eine Wurzel k_r.
Mit Wahrscheinlichkeit $w_{i,j}$ wird der gesuchte Schlüssel mit k_r verglichen.
Im Mittel werden $e_{i,r-1}$ Vergleiche im linken und $e_{r+1,j}$ im rechten Unterbaum durchgeführt. □

Wie berechnen wir $e_{i,j}$?

Lemma

- $e_{i,j} = 0$ *für* $i > j$
- $e_{i,j} = \min\limits_{i \leq r \leq j} (e_{i,r-1} + e_{r+1,j}) + w_{i,j}$ *für* $i \leq j$

Beweis.

Ein Baum mit den Knoten $k_i, \dots, k_j$ hat eine Wurzel k_r.
Mit Wahrscheinlichkeit $w_{i,j}$ wird der gesuchte Schlüssel mit k_r verglichen.
Im Mittel werden $e_{i,r-1}$ Vergleiche im linken und $e_{r+1,j}$ im rechten Unterbaum durchgeführt. $\square$

Wie berechnen wir $e_{i,j}$?

Lemma

- $e_{i,j} = 0$ *für* $i > j$
- $e_{i,j} = \min_{i \le r \le j} (e_{i,r-1} + e_{r+1,j}) + w_{i,j}$ *für* $i \le j$

Beweis.

Ein Baum mit den Knoten $k_i, \ldots, k_j$ hat eine Wurzel k_r.
Mit Wahrscheinlichkeit $w_{i,j}$ wird der gesuchte Schlüssel mit k_r verglichen.
Im Mittel werden $e_{i,r-1}$ Vergleiche im linken und $e_{r+1,j}$ im rechten Unterbaum durchgeführt. □

Wie berechnen wir $e_{i,j}$?

Lemma

- $e_{i,j} = 0$ *für* $i > j$
- $e_{i,j} = \min\limits_{i \leq r \leq j} (e_{i,r-1} + e_{r+1,j}) + w_{i,j}$ *für* $i \leq j$

Beweis.

Ein Baum mit den Knoten $k_i, \ldots, k_j$ hat eine Wurzel k_r.
Mit Wahrscheinlichkeit $w_{i,j}$ wird der gesuchte Schlüssel mit k_r verglichen.
Im Mittel werden $e_{i,r-1}$ Vergleiche im linken und $e_{r+1,j}$ im rechten Unterbaum durchgeführt. $\square$

i	1	2	3	4	5
p_i	$1/4$	$1/8$	$1/6$	$1/4$	$5/24$

w	1	2	3	4	5
1	0.25	0.375	0.542	0.792	1.0
2	0.0	0.125	0.292	0.542	0.75
3	0.0	0.0	0.167	0.417	0.625
4	0.0	0.0	0.0	0.25	0.458
5	0.0	0.0	0.0	0.0	0.208

e	1	2	3	4	5
1	0.25	0.5	0.958	1.542	2.167
2	0.0	0.125	0.417	0.917	1.375
3	0.0	0.0	0.167	0.583	1.0
4	0.0	0.0	0.0	0.25	0.667
5	0.0	0.0	0.0	0.0	0.208

$$e_{i,j} = \min_{i \leq r \leq j} (e_{i,r-1} + e_{r+1,j}) + w_{i,j}$$

Resultierender Algorithmus:

Algorithmus

for $i = 1, \ldots, n$ **do** $e[i, i-1] := 0$ **od**;
for $l = 0, \ldots, n$ **do**
  for $i = 1, \ldots, n-l$ **do**
    $j := i + l$;
    $e[i, j] := \text{infty}$;
    for $r = i, \ldots, j$ **do**
      $e[i, j] := \min\{\, e[i, j], e[i, r-1] + e[r+1, j] + w[i, j]\}$
    od
  od
od

Laufzeit: $O(n^3)$

Optimale Suchbäume – Allgemeiner Fall

- Es seien $k_1 < \cdots < k_n$ Schlüssel.
- Wir suchen k_i mit Wahrscheinlichkeit p_i.
- Wir suchen zwischen k_i und k_{i+1} mit Wahrscheinlichkeit q_i.
- (Mit W'keit q_0 wird links von k_1 gesucht.)
- (Mit W'keit q_n wird rechts von k_n gesucht.)

Natürlich gelte

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

Optimale Suchbäume – Allgemeiner Fall

Definition

Ein optimaler Suchbaum für $k_1, \ldots, k_n$ gemäß den Zugriffswahrscheinlichkeiten $p_1, \ldots, p_n$ und $q_0, \ldots, q_n$ ist

- ein Suchbaum für $k_1, \ldots, k_n$,
- und hat eine minimale Anzahl von Vergleichen im Erwartungswert, falls k_i mit Wahrscheinlichkeit p_i und zwischen k_i und k_{i+1} mit Wahrscheinlichkeit q_i gesucht wird.

Java

```
public void opt_searchtree(int n, List⟨K⟩ keys, List⟨Double⟩ p, List⟨Double⟩ q) {
  double[][] e = new double[n + 2][n + 1];
  double[][] w = new double[n + 2][n + 1];
  int[][] root = new int[n + 2][n + 1];
  for(int i = 1; i ≤ n + 1; i++) w[i][i − 1] = e[i][i − 1] = q.get(i − 1);
  for(int l = 0; l ≤ n; l++)
    for(int i = 1; i + l ≤ n; i++) {
      e[i][i + l] = Double.MAX_VALUE;
      w[i][i + l] = w[i][i + l − 1] + p.get(i + l) + q.get(i + l);
      for(int r = i; r ≤ i + l; r++) {
        Double t = e[i][r − 1] + e[r + 1][i + l] + w[i][i + l];
        if(t < e[i][i + l]) { e[i][i + l] = t; root[i][i + l] = r; }
      }
    }
  construct_opt_tree(1, n, keys, root);
}
```

Konstruktion optimaler Suchbäume

Java

```
void construct_opt_tree(int i, int j, List<K> keys, int[][] root) {
  if(j < i) return;
  int r = root[i][j];
  insert(keys.get(r), null);
  construct_opt_tree(i, r - 1, keys, root);
  construct_opt_tree(r + 1, j, keys, root);
}
```

Konstruktion optimaler Suchbäume

Theorem

Wir können einen optimalen Suchbaum für n Schlüssel in $O(n^3)$ Schritten konstruieren.