

Programmieren / Algorithmen & Datenstrukturen 2

Fortgeschrittenes Suchen

Prof. Dr. Skroch



Universitatea
BABEŞ-BOLYAI

Fortgeschrittenes Suchen

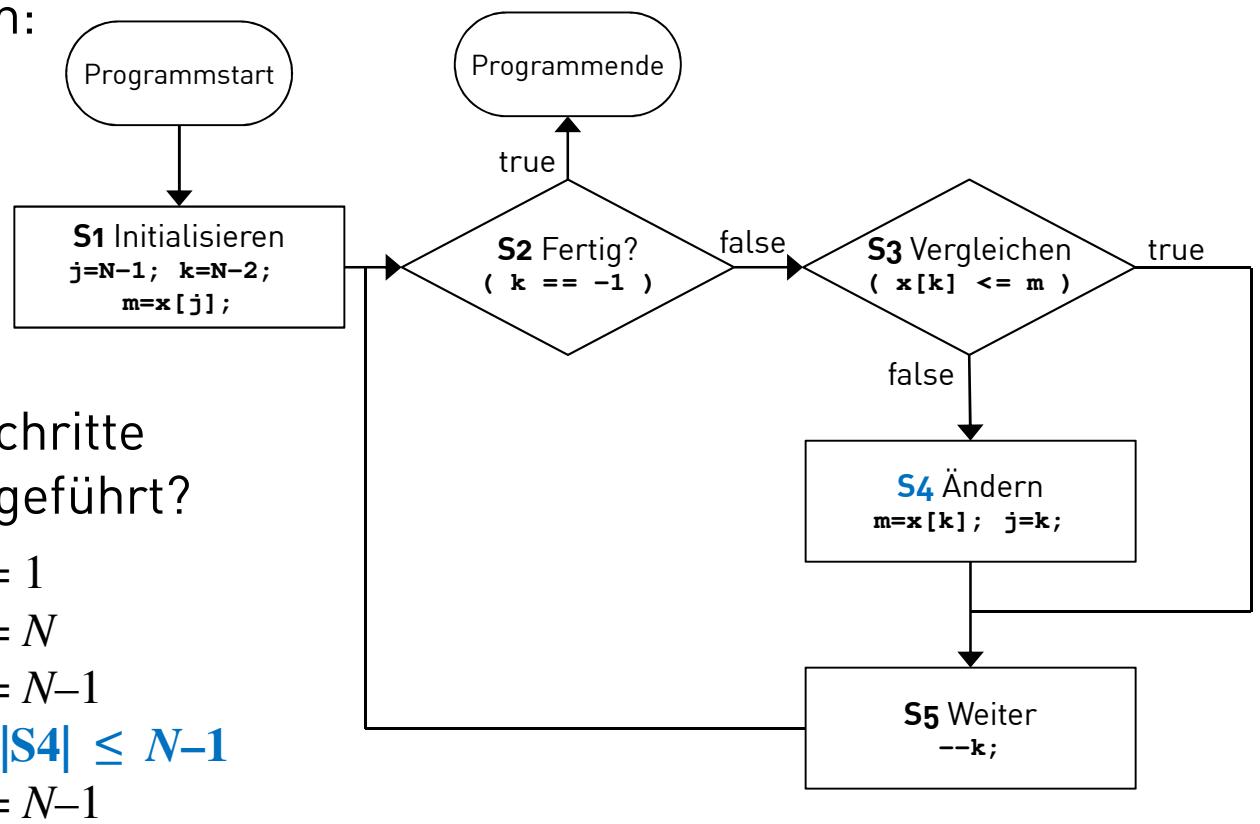
Inhalt.

- ▶ Templates
- ▶ Abgeleitete Klassen
- ▶ Testgetriebene Programmierung
- ▶ Container, Iteratoren und Algorithmen der StdLib
- ▶ Fortgeschrittenes Suchen
- ▶ Fortgeschrittenes Sortieren
- ▶ Grafische Benutzeroberflächen

Nochmal die sequentielle Suche

Ein genauer Blick auf eine einfache, sequentiellen Suche, die in N unterschiedlichen Elementen das größte Element sucht.

- ▶ Gegeben seien N Elemente x_0, x_1, \dots, x_{N-1} ($N \geq 1$).
- ▶ Gesucht seien m und j so, dass $m = x_j = \max_{0 \leq i \leq (N-1)} (x_i)$.
- ▶ Programmablaufplan:



- ▶ Wie oft werden die Schritte S1 bis S5 jeweils ausgeführt?

$$\begin{aligned}1 \text{ mal S1: } & |S1| = 1 \\N \text{ mal S2: } & |S2| = N \\N-1 \text{ mal S3: } & |S3| = N-1 \\? \text{ mal S4: } & \textcolor{blue}{0 \leq |S4| \leq N-1} \\N-1 \text{ mal S5: } & |S5| = N-1\end{aligned}$$

Nochmal die sequentielle Suche

Ein genauer Blick auf eine einfache, sequentiellen Suche, die in N unterschiedlichen Elementen das größte Element sucht.

► Wie oft Schritt S4 ausgeführt wird, hängt von der Reihenfolge der x_i ab.

- Günstigster Fall: $|S4| = 0$; falls $x_{N-1} = \max_{0 \leq i \leq (N-1)} (x_i)$;
- Ungünstigster Fall: $|S4| = N-1$; falls $x_0 > x_1 > \dots > x_{N-1}$;

► Durchschnittlicher Fall?

- Annahme zu den zu durchsuchenden Daten:
 - Die x_i sind lauter unterschiedliche Werte,
 - jede Permutation dieser Werte ist gleich wahrscheinlich.
- Beispielsweise sind für $N=3$ Elemente folgende sechs Fälle möglich:

$$x_0 < x_1 < x_2 : |S4| = 0; \quad x_0 < x_2 < x_1 : |S4| = 1;$$

$$x_1 < x_0 < x_2 : |S4| = 0; \quad x_1 < x_2 < x_0 : |S4| = 1;$$

$$x_2 < x_0 < x_1 : |S4| = 1; \quad x_2 < x_1 < x_0 : |S4| = 2;$$

Nochmal die sequentielle Suche

Ein genauer Blick auf eine einfache, sequentiellen Suche, die in N unterschiedlichen Elementen das größte Element sucht.

► Durchschnittlicher Fall allgemein:

- Die *Wahrscheinlichkeit*, dass S_4 für N Elemente genau k mal ausgeführt wird, ist

$$p_{N,k} = \frac{\text{Anzahl der Permutationen, für die } |S_4| = k \text{ ist}}{N!};$$

Beispielsweise ist für $N = 3$ Elemente $p_{3,0} = 1/3$; $p_{3,1} = 1/2$; $p_{3,2} = 1/6$;

- Der *Erwartungswert* für die Anzahl der Änderungen $|S_4|$ bei N Elementen ist

$$E_N = \sum_k kp_{N,k};$$

Beispielsweise ist $E_3 = 5/6$

- Wie können die Erwartungswerte E_N berechnet werden, ohne die Anzahl der entsprechenden Permutationen für jeden Fall manuell zu kombinieren?
- Die Frage wird schnell recht kompliziert...

Nochmal die sequentielle Suche

Ein genauer Blick auf eine einfache, sequentiellen Suche, die in N unterschiedlichen Elementen das größte Element sucht.

- Zur Berechnung ist folgende induktive Überlegung hilfreich
 - Wir betrachten zur Vereinfachung (und o.B.d.A.) alle Permutationen x_1, x_2, \dots, x_N auf $\{1, 2, \dots, N\}$
 - Ist $x_1 = N$, so braucht man für x_1, x_2, \dots, x_N *genau ein S4 mehr* als für x_2, x_3, \dots, x_N
 - Ist $x_1 \neq N$, so braucht man für x_1, x_2, \dots, x_N *genau gleich oft S4* wie für x_2, x_3, \dots, x_N
- Wir bezeichnen mit $|S4|_{N,k}$ die Anzahl der Fälle, in denen beim Durchlauf durch N Elemente k mal Schritt S4 erforderlich wird
 - Jetzt lässt sich ableiten: $|S4|_{N,k} = |S4|_{(N-1),(k-1)} + (N-1)|S4|_{(N-1),k}$;
Bzw. gleichbedeutend: $p_{N,k} = \frac{1}{N} p_{(N-1),(k-1)} + \frac{N-1}{N} p_{(N-1),k}$;
 - Für die Induktion gelten noch die "Startbedingungen":
$$p_{1,k} = \begin{cases} 1 & \text{für } k = 0; \\ 0 & \text{für } k \neq 0; \end{cases} \quad p_{N,k} = 0 \quad \text{für } k < 0;$$

Nochmal die sequentielle Suche

Ein genauer Blick auf eine einfache, sequentiellen Suche, die in N unterschiedlichen Elementen das größte Element sucht.

- Man ist zu der einfachen Formel $E_N = \sum_k kp_{N,k} = H_N - 1$ gelangt.

- Dabei ist H_N eine sog. harmonische Zahl:

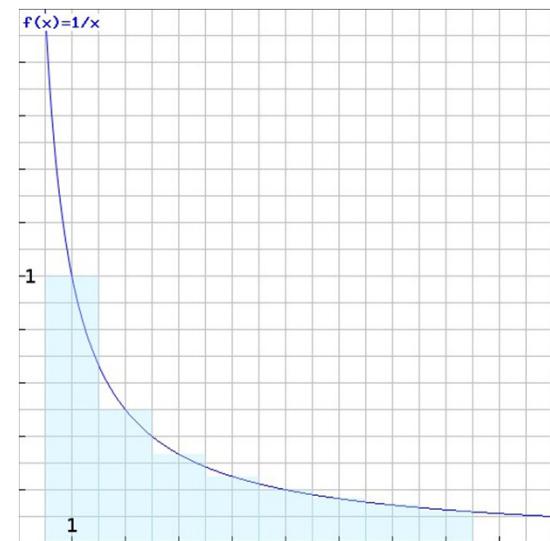
$$H_N = \sum_{k=1}^N \frac{1}{k} = 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{N};$$

- D.h. die N -te Teilsumme der harmonischen Reihe.
- Man kann sich harmonische Zahlen als diskrete Version des natürlichen Logarithmus vorstellen.
 - $\ln(N)$ entspricht der Fläche unter dem $1/x$ Graphen zwischen 1 und N .
 - H_N entspricht dieser Fläche, wenn man $1/x$ als diskrete "Treppenfunktion" nimmt.

- $\lim_{N \rightarrow \infty} H_N = \ln(N) + \gamma + O\left(\frac{1}{N}\right) \approx \ln(N); \quad \gamma \approx 0,5772$ ("Euler-Mascheroni-Konstante")

$$H_N < \ln(N) + 1 \text{ für } N \geq 2;$$

- Für beispielsweise $N = 12$ Elemente würde Schritt S4 also durchschnittlich $H_{12} - 1 \approx 2,103$ mal ausgeführt...



LEERE SEITE

Baum

Einige grundlegende Definitionen und Begriffe.

- ▶ Bäume sind gut untersuchte mathematische Strukturen.
 - Spezielle Graphen, deren Behandlung problemlos mehrere Bücher füllen könnte.
- ▶ Der Grundgedanke ist aus dem Alltag wohl vertraut.
 - Beispiel Familienstammbaum zur Darstellung der Verwandtschaftsbeziehungen von Sippen, Clans, u.ä.
 - Beispiel Verzeichnisbaum zur Organisation des Dateisystems durch Betriebssysteme für Computer.
- ▶ Ein Baum ist eine nichtleere Menge von Knoten und Kanten, mit bestimmten Eigenschaften:
 - Ein *Knoten* ist ein Objekt, das direkt identifizierbar ist (z.B. über einen Namen) und optional weitere Informationen enthalten kann (engl. *node* oder auch *vertex*),
 - eine *Kante* ist eine direkte Verbindungen zwischen zwei Knoten (engl. *edge*),
 - ein *Pfad* ist eine verzweigungsfreie Folge von unterschiedlichen Knoten, wobei direkt aufeinanderfolgende Knoten der Folge jeweils durch eine Kante verbunden sind.
- ▶ Die definierende Eigenschaft eines Baums kann so formuliert werden:
Es gibt genau einen Pfad, der zwei beliebige Knoten verbindet.

Wurzelbaum

Einige grundlegende Definitionen und Begriffe.

- ▶ Ein Wurzelbaum ist ein Baum, in dem einer der Knoten als *Wurzel* gekennzeichnet ist.
 - Andernfalls spricht man von einem freien Baum.
- ▶ Normalerweise stellt man sich vor, dass die Kanten entweder alle zur Wurzel hin oder alle von der Wurzel weg zeigen.
 - Hinweis: die Definition eines Baums besagt *nicht*, dass die Kanten gerichtet sind.
- ▶ In der Informatik wird einen Wurzelbaum üblicherweise so dargestellt, dass sich die Wurzel in der bildlichen Darstellung *oben* befindet.
 - Man sagt:
Knoten y liegt *unter* bzw. *nach* Knoten x
und x damit *über* bzw. *vor* y ,
wenn x auf dem Pfad von y zur Wurzel liegt.

Wurzelbaum

Einige grundlegende Definitionen und Begriffe.

- ▶ Jeder Knoten bildet die Wurzel für einen *Teilbaum* (oder *Unterbaum*).
 - Der Teilbaum besteht aus dem Knoten selbst und allen darunter liegenden Knoten.
- ▶ Jeder Knoten, außer der Wurzel, hat genau einen Knoten *direkt* über sich.
 - Dieser wird als *Vorgänger* des Knotens bezeichnet.
- ▶ Die Knoten *direkt* unter einem Knoten heißen *Nachfolger* des Knotens.
- ▶ Knoten ohne Nachfolger heißen *Blätter* (oder *Endknoten*).

Wurzelbaum

Einige grundlegende Definitionen und Begriffe.

► *Knotenebene:*

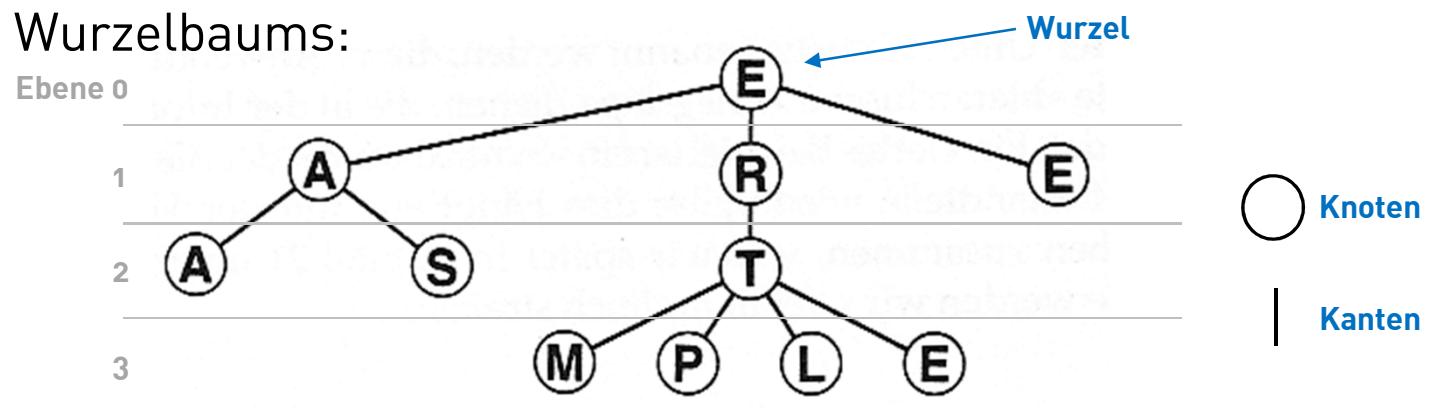
- Die Ebene der Wurzel ist 0 und für alle anderen Knoten gilt: die Ebene ist jeweils um 1 höher als die des Vorgängers.
- Die Ebene eines Knotens wird auch *Abstand vom Knoten zur Wurzel* genannt.

► Die Summe der Ebenen aller Knoten wird als *Pfadlänge* des Wurzelbaums bezeichnet.

► *Höhe* eines Wurzelbaums.

- Der maximale Abstand zwischen irgend einem Knoten und der Wurzel.
- Oder gleichbedeutend: die höchste Ebene, die unter allen Knoten auftritt.

► Beispiel eines Wurzelbaums:



Geordneter Baum, M -wertiger Baum

Einige grundlegende Definitionen und Begriffe.

► Geordneter Baum:

- Ein Wurzelbaum, in dem die Reihenfolge der Nachfolger an jedem Knoten systematisch festgelegt ist.
- Eine geeignet gewählte Systematik ermöglicht bestimmte, wünschenswerte Strukturen des geordneten Baums.
 - In einer Weise, die für die beabsichtigte Anwendung des geordneten Baums möglichst vorteilhaft ist.

► M -wertiger Baum:

- Ein Wurzelbaum, in dem jeder Knoten eine bestimmte Anzahl M von Nachfolgern haben *muss*.
- In M -wertigen Bäumen gibt es deshalb spezielle Knoten, die als *externe Knoten* oder *Nullknoten* bezeichnet werden.
 - Grund: Knoten, die in einer Instanz der M -wertigen Baumstruktur weniger als M "echte" Nachfolger haben, erhalten entsprechend externe Knoten als weitere Nachfolger.
- Kanten zu externen Knoten heißen auch *Nullkanten* oder *Nullverbindungen*.
- Ein Blatt in einem M -wertigen Baum hat *nur* externe Knoten als Nachfolger.

Binärbaum

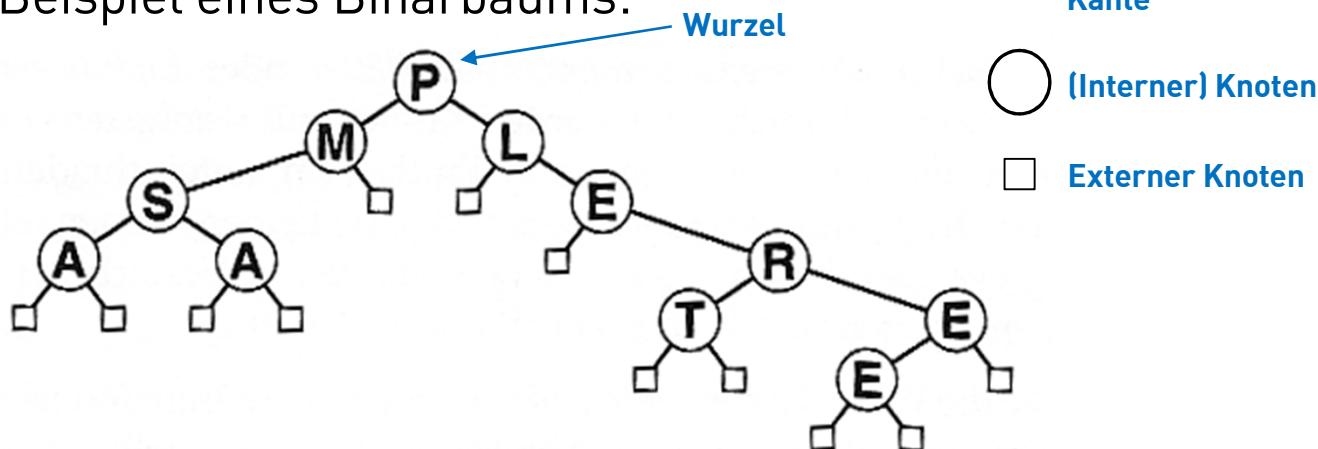
Einige grundlegende Definitionen und Begriffe.

- ▶ Ein Binärbaum ist ein geordneter, 2-wertiger Baum.
 - Ein Binärbaum besteht aus zwei Typen von Knoten:
 - externe Knoten: Knoten ohne Nachfolger.
 - interne Knoten, auch einfach nur "Knoten" genannt:
Knoten mit genau zwei Nachfolgern.
 - Da die beiden Nachfolger eines Knotens geordnet sind, werden sie zur eindeutigen Unterscheidung als *linker Nachfolger* und *rechter Nachfolger* bezeichnet.
 - Jeder interne Knoten *muss* sowohl einen linken als auch einen rechten Nachfolger haben.
 - Einer oder beide von ihnen können externe Knoten sein.
 - Ein Blatt in einem Binärbaum hat zwei externe Knoten als Nachfolger.
- ▶ Der Zweck eines Binärbaums ist die Strukturierung der internen Knoten, die externen Knoten dienen nur als Platzhalter.
- ▶ Die *externe* bzw. *interne Pfadlänge* eines Binärbaums ist die Summe der Ebenen aller seiner externen bzw. internen Knoten.

Binärbaum

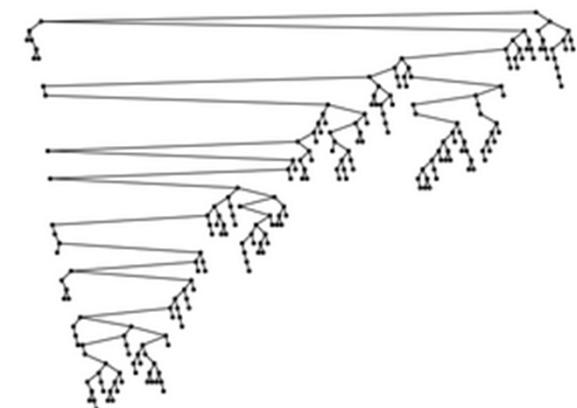
Beispiel und einfache Implementierung.

- Beispiel eines Binärbaums:



- Ein Binärbaum ist

- entweder ein externer Knoten
- oder ein interner Knoten, der mit einem geordneten Paar von Binärbäumen verbunden ist,
 - wobei man diese beiden Binärbäume als *linker Teilbaum* bzw. *rechter Teilbaum* des Knotens bezeichnet



Binärbaum

Beispiel und einfache Implementierung.

- ▶ Beispielskizze für eine einfache, mögliche Implementierung als Typ:

```
class BinTree {  
public:  
    BinTree() : root{'\0'}, sz{0} {}  
    ~BinTree();  
    int size() const { return sz; }  
    node* find( const char& val ) const;  
    node* insert( const char& val );  
    // ...  
  
private:  
    struct node {  
        char value;  
        node* top; // optional Zeiger auf Vorgaenger  
        node* left;  
        node* right;  
        explicit node( const char& ch, const node* t = nullptr )  
            : value{ch}, top{t}, left{nullptr}, right{nullptr} {}  
    };  
    node* root; // die Wurzel  
    int sz; // Anzahl der Knoten  
    // ...  
};
```

Binärbaum

Einige wichtige Eigenschaften von Binärbäumen.

- ▶ Ein Binärbaum mit N Knoten hat $N-1$ Kanten.
 - Anm.: diese Eigenschaft gilt für *alle Bäume*, nicht nur für Binärbäume.
 - ▶ Ein Binärbaum mit N internen Knoten hat $N+1$ externe Knoten.
 - ▶ Ein Binärbaum mit N internen Knoten hat $2N$ Kanten.
 - $N-1$ Kanten zwischen internen Knoten,
 - $N+1$ Kanten zu externen Knoten ("Nullverbindungen").
 - ▶ Die externe Pfadlänge $\pi_e(N)$ eines Binärbaums mit N internen Knoten ist $2N$ plus seine interne Pfadlänge $\pi_i(N)$, Begründung:
 - Es ist $\pi_i(1) = 0$ und $\pi_e(1) = 2$.
 - Jeder zusätzliche interne Knoten
 - erhöht N um 1,
 - erhöht π_i um m ,
 - und erhöht π_e um $2(m+1) = 2m + 2$,
- wobei m die Ebene ist, auf der der zusätzliche interne Knoten dazukommt.

Binärbaum

Einige wichtige Eigenschaften von Binärbäumen.

► Die Höhe eines Binärbaums mit N (internen) Knoten beträgt

- höchstens $N-1$
 - (beim sog. entarteten Baum:
 - auf jeder Ebene befindet sich genau ein interner Knoten,
 - der Knoten auf der letzten Ebene (Baumhöhe) hat keinen internen Nachfolger,
 - die übrigen Knoten haben jeweils genau einen internen Nachfolger)
- und mindestens $\lfloor \log_2 N \rfloor$
 - (beim sog. ausgeglichenen Baum:
 - auf jeder Ebene m befinden sich genau 2^m interne Knoten, bis auf die letzte Ebene (Baumhöhe), auf der sich maximal 2^m interne Knoten befinden,
 - die Knoten auf der letzten Ebene (Baumhöhe) haben keine internen Nachfolger,
 - die Knoten auf der vorletzten Ebene haben jeweils höchstens zwei interne Nachfolger,
 - die übrigen Knoten haben jeweils genau zwei interne Nachfolger).

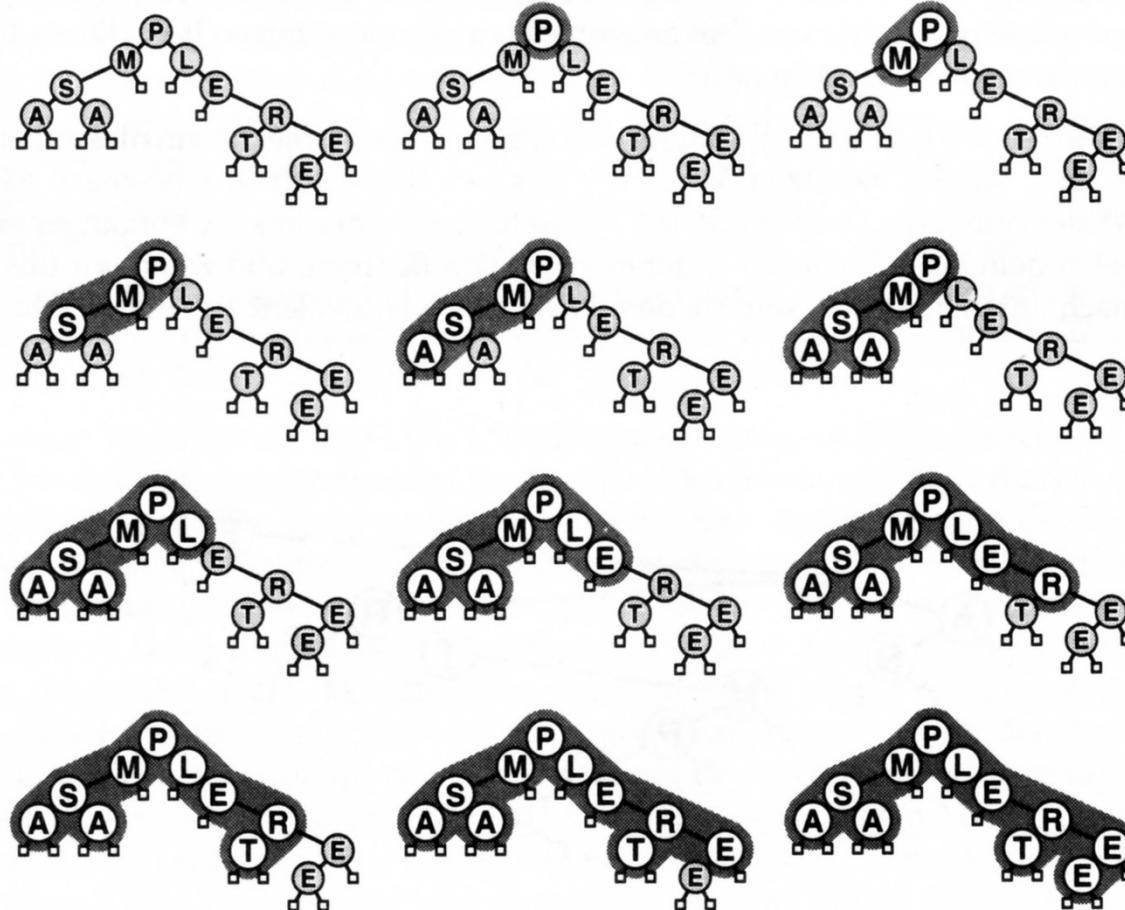
Binärbaum

Traversierung.

- ▶ Traversierung eines Binärbaums bedeutet, dass man systematisch alle seine Knoten in einer bestimmten Reihenfolge betrachtet (man sagt auch "besucht").
- ▶ Es gibt unterschiedlichen Traversierungsmethoden, die sich vor allem in der Reihenfolge unterscheiden, in der die Knoten betrachtet werden.
 - *Preorder*: besucht erst die Wurzel, dann rekursiv linke Teilbäume und zuletzt rekursiv rechte Teilbäume.
 - *Inorder*: besucht rekursiv erst linke Teilbäume, dann die Wurzel und zuletzt rekursiv rechte Teilbäume.
 - *Postorder*: besucht rekursiv erst linke Teilbäume, dann rekursiv rechte Teilbäume und zuletzt die Wurzel.
 - *Level-order*: besucht erst die Wurzel des Baums, wechselt dann auf die nächsthöhere Knotenebene 1 und besucht alle Knoten der Ebene von links nach rechts, wechselt dann auf die nächsthöhere Knotenebene 2, usw. bis zur Höhe des Baums.

Binärbaum

Preorder Traversierung.

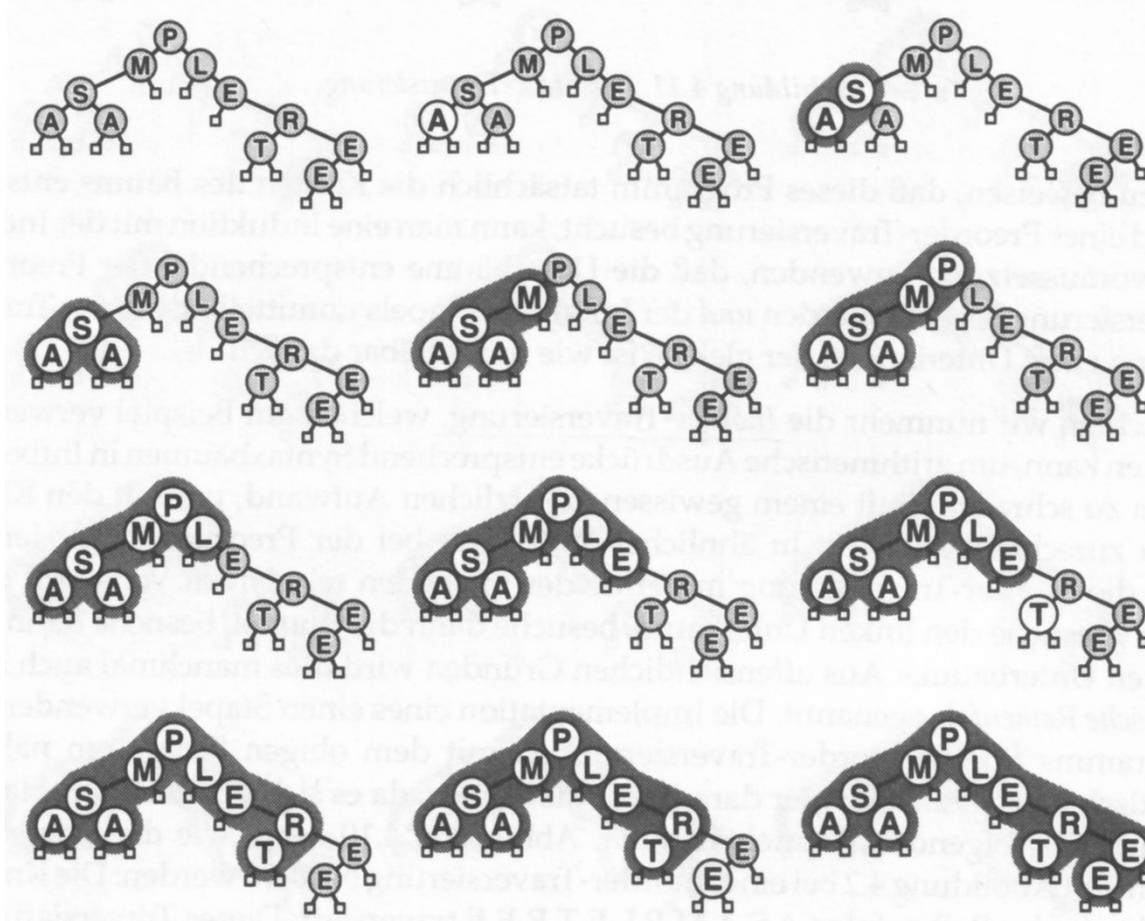


PMSAALERTEE

```
void pre_order( node* n ) {  
    if( n==nullptr ) return;  
    visit( n ); // z.B. cout  
    if( n->left != nullptr )  
        pre_order( n->left );  
    if( n->right != nullptr )  
        pre_order( n->right );  
}
```

Binärbaum

Inorder Traversierung.

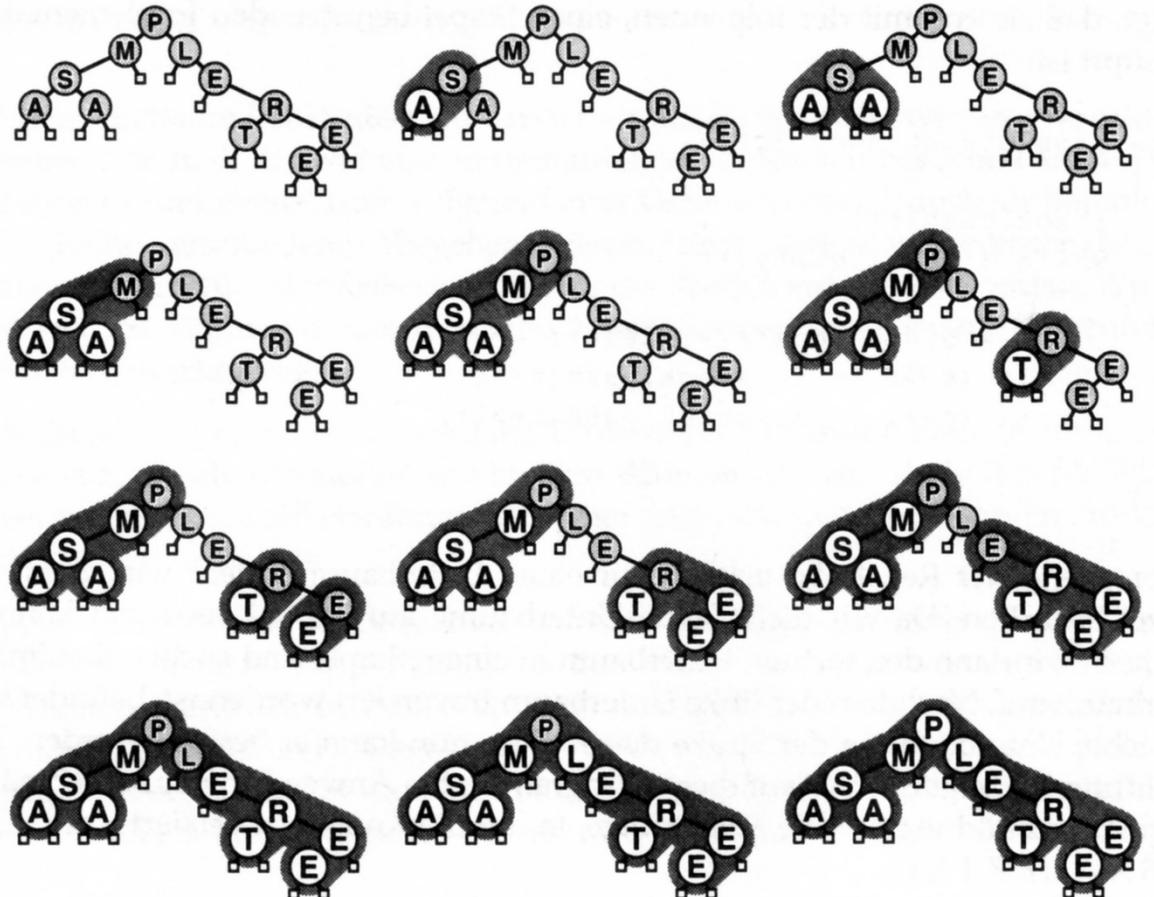


ASAMPLETREE

```
void in_order( node* n ) {  
    if (n != NULL) {  
        in_order(n->left);  
        cout << n->data;  
        in_order(n->right);  
    }  
}
```

Binärbaum

Postorder Traversierung.



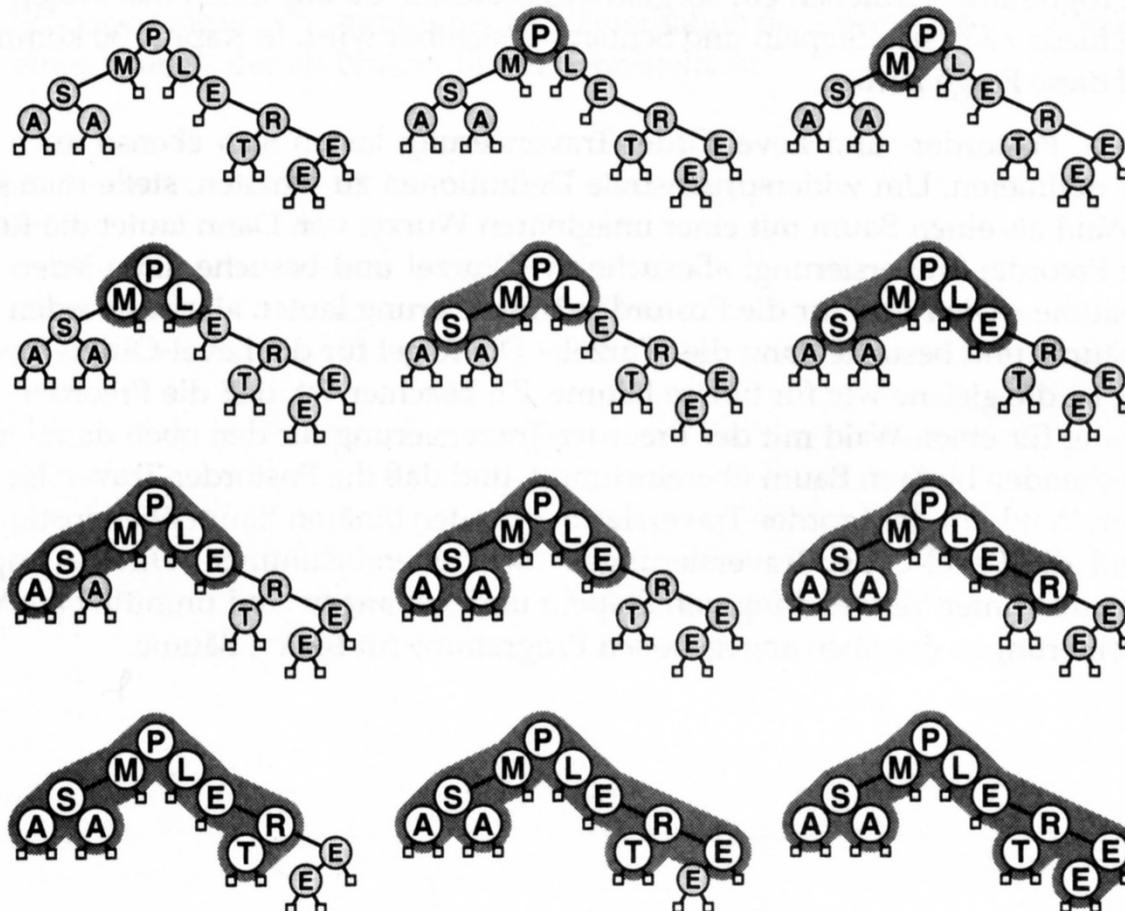
AASMTEERELP

```
void post_order( node* n ) {
```

```
}
```

Binärbaum

Level-order Traversierung.



PMLSEAARTEE

```
void level_order( node* n ) {  
    if( n==nullptr ) return;  
    std::queue<node*> q{};  
    q.push( n );  
    while( !q.empty() ) [  
        n = q.front();  
        q.pop();  
        visit( n ); // z.B. cout  
        if( n->left != nullptr )  
            q.push( n->left );  
        if( n->right != nullptr )  
            q.push( n->right );  
    ]  
}
```

Binärer Suchbaum

Definition.

- ▶ Ein binärer Suchbaum (BST, binary search tree) ist ein Binärbaum, dessen Anwendungszweck darin besteht, dass nach Schlüsselwerten gesucht werden soll.
- ▶ In einem BST ist daher folgende Ordnungs-Systematik festgelegt:
 - Von jedem Knoten aus gesehen sind alle Schlüssel im linken Teilbaum kleiner als der eigene Schlüssel.
 - Von jedem Knoten aus gesehen ist der eigene Schlüssel kleiner oder gleich aller Schlüssel im rechten Teilbaum.
- ▶ Oft haben Knoten im BST neben dem Schlüsselwert einen zweiten Wert:

```
template<class T1, class T2> struct node {  
    T1 key;  
    T2 value;  
    node* top; // optional  
    node* left;  
    node* right;  
    //...  
};
```

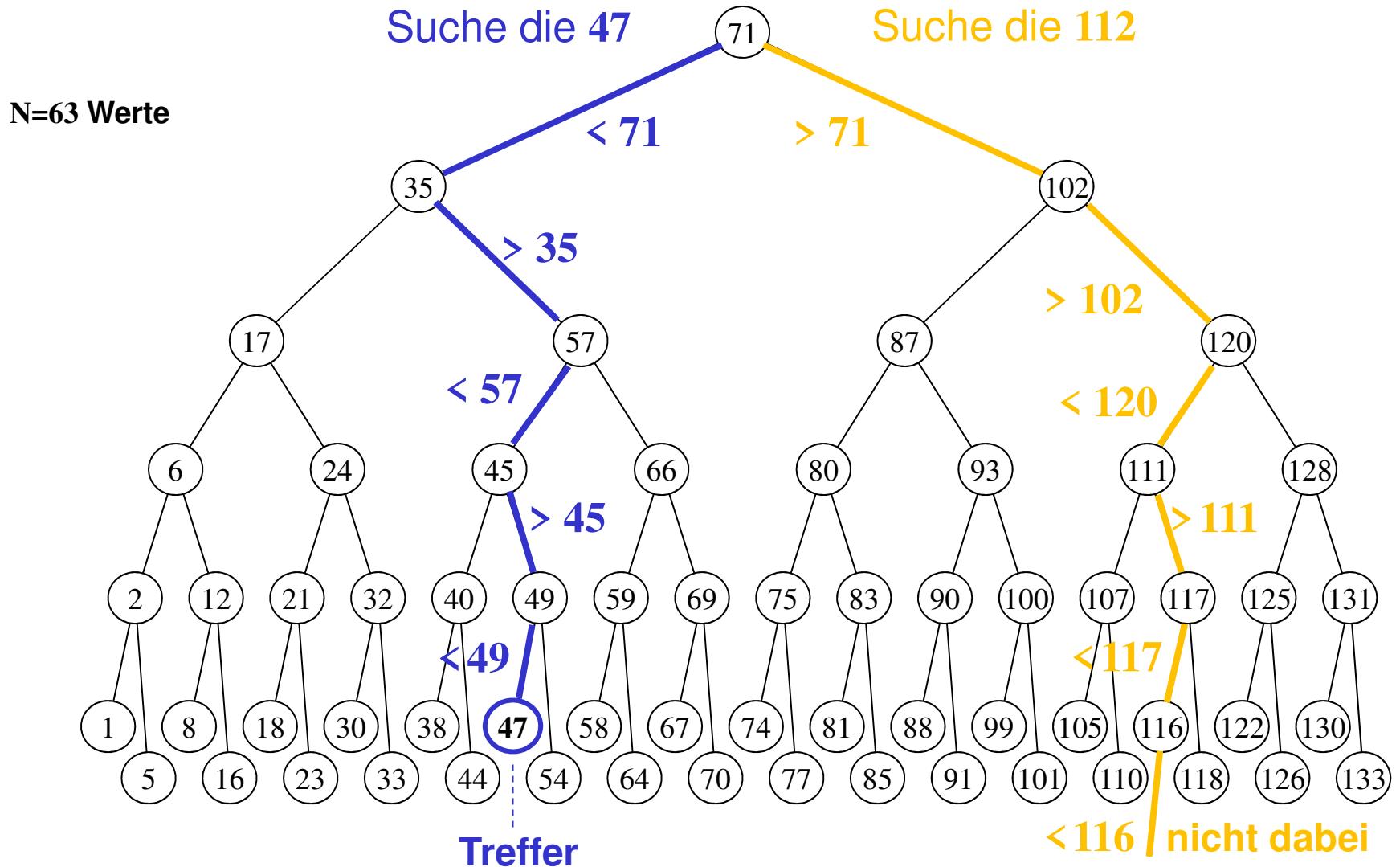
Binärer Suchbaum

Suchen.

- ▶ Die *Suche* nach einem Schlüsselwert in einem BST ist aufgrund seines sortierten Aufbaus einfach.
 - Die Suche beginnt bei der Wurzel.
 - Wenn die Wurzel den gesuchten Schlüsselwert enthält, liegt ein Suchtreffer vor.
 - Wenn der gesuchte Schlüsselwert *kleiner* als der Schlüsselwert der Wurzel ist, wird die Suche rekursiv im *linken Teilbaum* fortgesetzt.
 - Ansonsten wird die Suche rekursiv im *rechten Teilbaum* fortgesetzt.
 - Entweder wird der Schlüsselwert an der Wurzel eines Teilbaums gefunden (Suchtreffer).
 - Oder an der Stelle, wo die Suche fortgesetzt werden müsste, taucht eine Nullverbindung auf, womit der Schlüsselwert nicht im BST vorhanden ist (erfolglose Suche).
- ▶ Implementierung in $O(h)$ Laufzeit mit h = Höhe des BST,
 - wenn Schlüsselvergleiche und Zeigeroperationen in $O(1)$ konstanter Zeit möglich sind.

Binärer Suchbaum

Ein Beispiel zur Veranschaulichung (im Bild ein sog. *vollständiger* BST).



Binärer Suchbaum

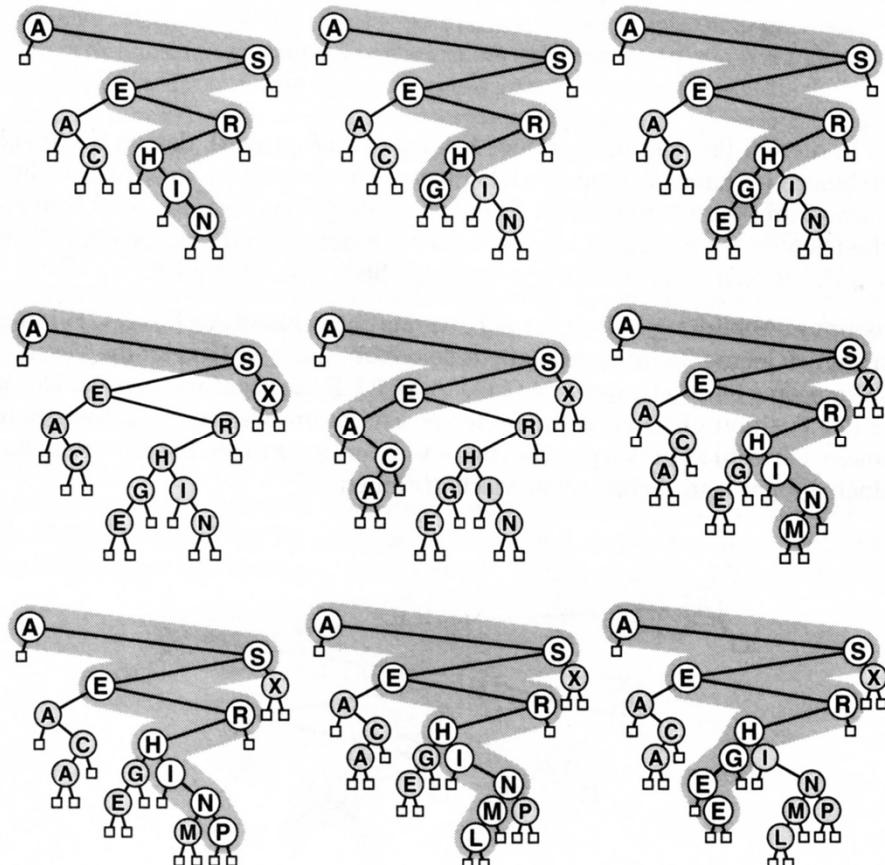
Einfügen.

- Das *Einfügen* eines neuen Schlüsselwerts in einen BST kann der selben Logik wie die Suche folgen und ist dann genauso einfach.
 - Einen neuen Knoten kann man einfügen, indem man zunächst durch eine erfolglose Suche (die in einer Nullverbindung endet) die Stelle findet, an welcher der Wert einzufügen ist.
 - Nun kann man die Nullverbindung durch einen Zeiger auf den neu einzufügenden Knoten ersetzen.
 - Sind gleiche Schlüsselwerte mehrfach erlaubt, fällt der neue Knoten rechts neben bereits vorhandene Knoten, die schon diesen Schlüsselwert haben.
 - D.h. es wird nach einem Suchtreffer einfach die Suche solange weiter fortgesetzt, bis eine Nullverbindung auftaucht, worauf diese durch einen Zeiger auf den neuen Knoten ersetzt wird.
 - Eine je nach Anwendungszweck des BST geeignetere Alternative kann darin bestehen, bei mehrfachen Schlüsselwerten nicht neue Knoten zu erzeugen, sondern in jedem Knoten zusätzlich zu speichern, wie oft der Schlüssel eingefügt wurde.

Binärer Suchbaum

Konstruktion eines gültigen binären Suchbaums, sortierte Ausgabe der Knoten.

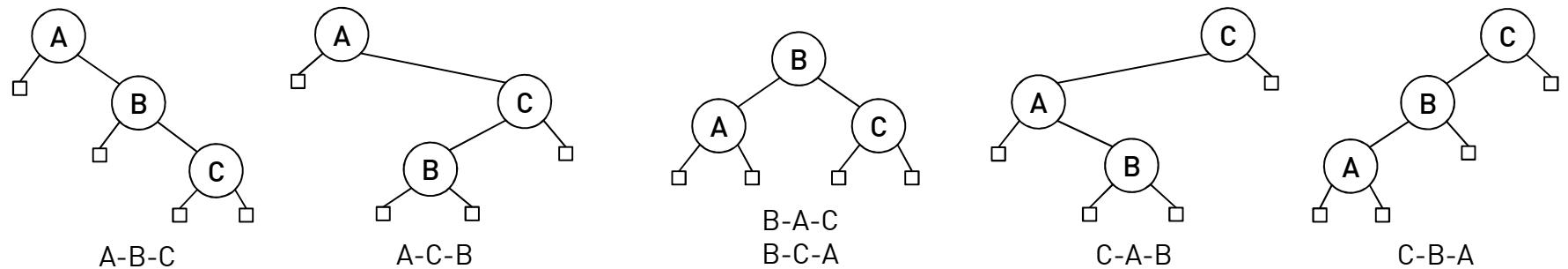
- ▶ Ein gültiger BST kann dadurch konstruiert werden, dass nacheinander neue Knoten nach der eben vorgestellten Methode eingefügt werden.
- ▶ Beispiel
 - In einem leeren BST wurden zunächst nacheinander die Schlüsselwerte A, S, E, A, R, C, H und I eingefügt.
 - Die Abbildung rechts zeigt, wie sich dieser BST weiter aufbaut, wenn nun weitere Schlüsselwerte N, G, E, X, A, M, P, L, E nacheinander eingefügt werden.
- ▶ Durch *Inorder-Traversierung* werden die Knoten in einem BST nach aufsteigenden Schlüsselwerten sortiert ausgegeben.



Binärer Suchbaum

Einige wichtige BST-Eigenschaften.

- ▶ Die Struktur einer BST-Instanz hängt wesentlich von der Reihenfolge der Einfügungen (und ggf. Löschungen) ab.
 - N Werte können in $N!$ unterschiedlichen Reihenfolgen eingefügt werden,
 - es gibt aber weniger als $N!$ unterschiedliche BSTs mit N Knoten.
- Beispiel, es gibt $5 < 3!$ unterschiedliche BSTs mit $N = 3$ Knoten:



- ▶ Die Struktur einer BST-Instanz wird also von den Invarianten des BST *und* von den Eigenschaften von *Permutationen* bestimmt.

Bemerkung: es gibt hierzu unterschiedliche Denkmodelle...

- ▶ *BST-Modell*: jedes der N Elemente ist mit gleicher Wkt. von $1/N$ die Wurzel (i.Allg. in der Informatik, wie oben).
- ▶ *Catalan-Modell*: jede Form des Binärbaums ist gleich wahrscheinlich (i.Allg. in der Mathematik).

Binärer Suchbaum

Einige wichtige BST-Eigenschaften.

- ▶ Die Suche in einem BST mit N Schlüsselwerten kann im *ungünstigsten Fall* N Vergleiche benötigen.
 - Wenn die Schlüsselwerte in geordneter Reihenfolge eingefügt werden, entsteht ein sog. *entarteter BST*, der die Höhe $N-1$ und die Struktur einer verketteten Liste hat (die Suche entspricht dann der sequentielle Suche).
- ▶ Die Anzahl der Vergleiche für einen Suchtreffer an einem Knoten im BST ist:
 - eins größer als die Ebene (d.h. der Beitrag zur Pfadlänge) des Knotens,
 - eins größer als die Anzahl an Vergleichen, die erforderlich waren, um diesen Knoten in den Baum einzufügen.
- ▶ Wie viele Vergleiche werden bei der Suche im BST im durchschnittlichen Fall benötigt?
 - Der *durchschnittliche Suchfall* ist zu definieren...
 - Man muss auch klären, was man unter einem *durchschnittlichen BST* verstehen möchte...

Binärer Suchbaum

Suchtreffer: durchschnittliche Anzahl an Vergleichen.

- ▶ Ist jede Permutation der Einfügungsreihenfolge von N unterschiedlichen Schlüsselwerten gleich wahrscheinlich, so erfordern Suchtreffer, erfolglose Suchen und Einfügungen jeweils durchschnittlich ungefähr $1,39 \log_2 N \approx 2 \ln N$ Vergleiche.
- ▶ Erläuterung:
 - Die Anzahl der Vergleiche zum Auffinden eines Werts ist genau eins größer als die Anzahl der Vergleiche, die benötigt wurden, um diesen Wert in den BST einzufügen.
 - Sei C'_N die durchschnittliche Anzahl der Vergleiche für eine erfolglose Suche / Einfügung.
Annahme: jedes der $N+1$ Intervalle ohne Schlüsselwert im Baum ist mit gleicher Wahrscheinlichkeit von $1/(N+1)$ von der Suche bzw. Einfügung betroffen.
 - Sei C_N die durchschnittliche Anzahl der Vergleiche für einen Suchtreffer.
Annahme: nach jedem der N Werte im Baum wird mit gleicher Wahrscheinlichkeit von $1/N$ gesucht.
 - Es gilt damit $C'_N = \frac{\pi_e(N)}{N+1}$; $C_N = 1 + \frac{\pi_i(N)}{N} = 1 + \frac{C'_0 + C'_1 + \dots + C'_{N-1}}{N}$;
 - Wegen $\pi_e(N) = \pi_i(N) + 2N$ gilt (etwas unerwartet) auch:
$$C_N = \left(1 + \frac{1}{N}\right) C'_N - 1; \quad \text{T. Hibbard (1962)}$$
 - Die untere Gleichung oben eingesetzt, dann vereinfacht, ergibt die Differenzengleichung
$$(N+1)C'_N = 2N + C'_0 + C'_1 + \dots + C'_{N-1};$$

Binärer Suchbaum

Suchtreffer: durchschnittliche Anzahl an Vergleichen.

► Lösung:

$$(N+1)C'_N = 2N + C'_0 + C'_1 + \dots + C'_{N-1};$$

$$NC'_{N-1} = 2(N-1) + C'_0 + C'_1 + \dots + C'_{N-2}; \quad (N-1) \text{ für } N \text{ eingesetzt}$$

$$(N+1)C'_N - NC'_{N-1} = 2 + C'_{N-1}; \quad \text{subtrahiert}$$

$$C'_N = C'_{N-1} + \frac{2}{N+1}; \quad \text{umgeformt}$$

$$C'_{N-1} = C'_{N-2} + \frac{2}{N}; \quad C'_N = C'_{N-2} + \frac{2}{N} + \frac{2}{N+1};$$

$$C'_{N-2} = C'_{N-3} + \frac{2}{N-1}; \quad C'_N = C'_{N-3} + \frac{2}{N-1} + \frac{2}{N} + \frac{2}{N+1}; \quad \text{usw.}$$

$$C'_N = 2 \cdot H_{N+1} - 2; \quad \text{minus 2 weil } C'_0 = 0 \text{ und } 2H_1 = 2$$

$$C_N = 2 \cdot \left(1 + \frac{1}{N}\right) H_N - 3; \quad \text{weil } C_N = \left(1 + \frac{1}{N}\right) C'_N - 1$$

$$C_N = 2H_N + \frac{2}{N}H_N - 3 \approx 2 \ln N; \quad \text{weil } H_N = \int_1^N \frac{1}{x} dx + \gamma \approx \int_1^N \frac{1}{x} dx;$$

$$\gamma \approx 0,5772 \quad (\text{"Euler-Mascheroni-Konstante"})$$

Binärer Suchbaum

Einfügen von neuen Knoten als Wurzel: durch rekursive Rotationsoperationen lässt sich ein solches Verfahren realisieren.

- ▶ In unserer bisherigen Methode zum Einfügen ersetzt ein neu eingefügter Knoten einen Nullknoten in der höchsten Ebene eines Teilbaums.
 - Das war nicht unbedingt so gefordert, sondern ist eine Eigenschaft des nahe liegenden Verfahrens zur Einfügung.
- ▶ Man kann als alternatives Verfahren fordern, dass Knoten immer als Wurzel eingefügt werden.
- ▶ Warum neue Knoten als Wurzel einfügen?
 - Der Suchaufwand für Knoten ist umso geringer, je näher sich die gesuchten Knoten an der Wurzel befinden (da die Suche immer in der Wurzel startet).
 - Man geht – aufgrund empirischer Erkenntnis – davon aus, dass sich in vielen Anwendungen die meisten Suchoperationen auf nur wenige Elemente beziehen, und dass dies häufig entweder die neu hinzugekommenen Elemente sind, oder solche Elemente, nach denen schon einmal gesucht wurde.
- ▶ Ändert man zusätzlich auch die Suchoperation so ab, dass bei jedem Suchtreffer der gefundene Knoten als neue Wurzel verwendet wird, hat man bereits ein *selbstorganisierendes Suchverfahren*, welches die häufig gesuchten Knoten nahe der Wurzel des BST sammelt...

Binärer Suchbaum

Operation: rotieren.

- ▶ Die Rotation vertauscht die Rolle der Wurzel eines Teilbaums und eines der beiden Nachfolger der Wurzel.
 - Dabei muss die Ordnung des BST bewahrt werden.

- ▶ Die *Rechtsrotation* wirkt auf die Wurzel und ihren linken Nachfolger.
 - Die Rechtsrotation bringt die Wurzel nach rechts und kehrt damit die Richtung ihrer linken Verbindung um: vor der Rotation zeigt sie von der Wurzel zum linken Nachfolger, nach der Rotation zeigt sie vom alten linken Nachfolger (der neuen Wurzel) auf die alte Wurzel (den rechten Nachfolger der neuen Wurzel) – der alte rechte Nachfolger der neuen Wurzel wird zum linken Nachfolger der ehemaligen Wurzel.
 - Mögliche Implementierung der Rechtsrotation:

```
using nodep = node*;
void rotateR( nodep& w ) {
```

```
// Übung
```

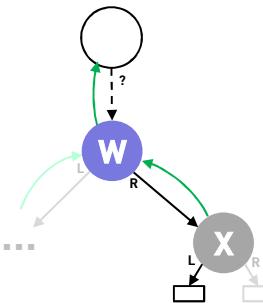
```
}
```

- ▶ Die *Linksrotation* wirkt analog auf eine Wurzel und ihren rechten Nachfolger.

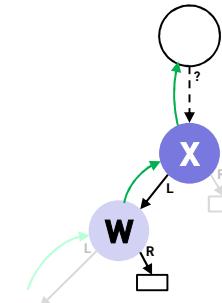
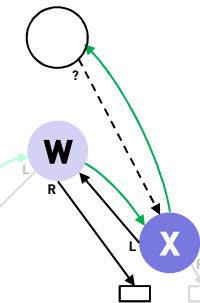
Binärer Suchbaum

Linksrotation (innere Knoten haben hier zusätzlich einen Zeiger auf ihren Vorgänger).

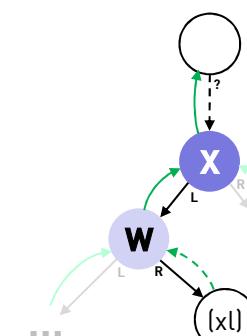
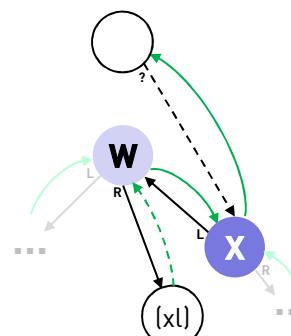
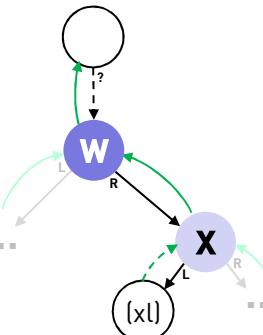
Einfügung von Blatt $X \geq W$



Schema der ersten Linksrotation



Schema weiterer Linksrotationen

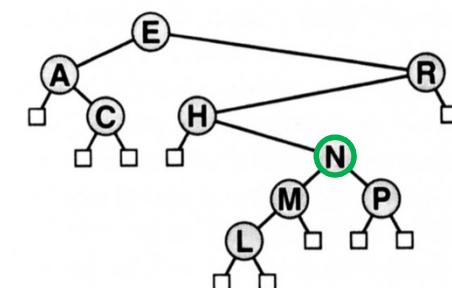
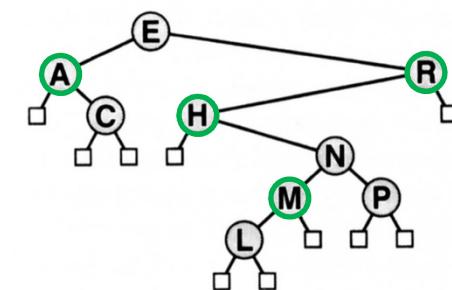
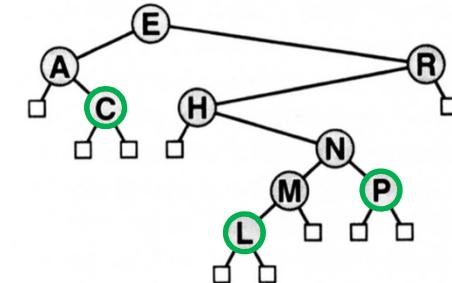


► Rechtsrotationen analog.

Binärer Suchbaum

Löschen von Knoten.

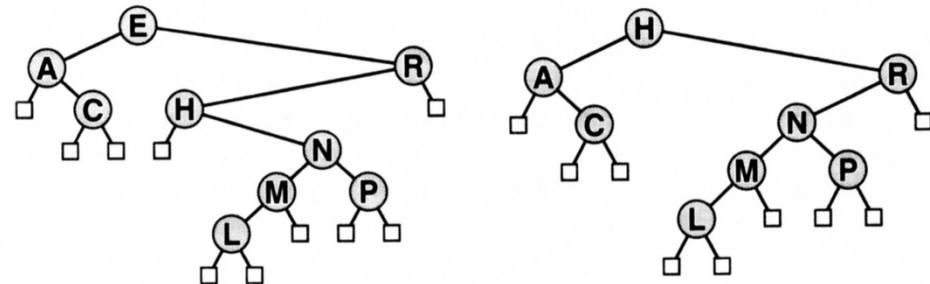
- ▶ Hat der zu löschende Knoten *zwei* externe Knoten als Nachfolger, wird einfach der Zeiger von seinem Vorgänger als Nullverbindung gesetzt und der aus dem BST entfernte Knoten gelöscht.
- ▶ Hat der zu löschende Knoten *einen* externen Knoten als Nachfolger, wird der Zeiger von seinem Vorgänger zum Zeiger auf seinen Nachfolger, worauf der aus dem BST entfernte Knoten gelöscht wird.
- ▶ Das Löschen von Knoten, die nicht mindestens einen Nullknoten als Nachfolger haben, bereitet mehr Schwierigkeiten.
 - Falls auf einen der beiden Nachfolger zwei externe Knoten folgen (wie bei N), kann man nochmals abkürzen.



Binärer Suchbaum

Löschen von Knoten, die nicht mindestens einen Nullknoten als Nachfolger haben.

- ▶ *Ansatz:* man ersetzt den zu löschenen Knoten durch den Knoten mit dem nächstgrößeren (oder nächstkleineren) Schlüsselwert.
 - Der nächstkleinere Knotenwert ist der größte Knotenwert im linken Teilbaum .
 - Der nächstgrößere Knotenwert ist der kleinste Knotenwert im rechten Teilbaum.
- ▶ Es gibt dann im BST keinen weiteren Schlüsselwert zwischen dem zu löschenen Knoten und dem Knoten, der ihn ersetzen soll.
 - D.h. dieser Knoten, der den zu löschenen Knoten ersetzen wird, hat mindestens einen externen Knoten als Nachfolger,
 - er kann daher leicht wie eben beschrieben aus dem BST entfernt werden
 - und kann direkt an Stelle des eigentlich zu löschenen Knotens treten.
 - Der Sonderfall gleichgroßer Knotenwerte kann ggf. einfach behandelt werden.
- ▶ Im Beispiel: der H-Knoten wird aus seiner Position im BST entfernt und tritt an Stelle des zu löschenen E-Knotens.



Binärer Suchbaum

Vertiefung: Operation "Finden des k -kleinsten Knotens".

- ▶ Suchen des kleinsten (größten) Knotens im rechten (linken) Teilbaum kann leicht implementiert werden: einmal nach rechts (links), dann immer nach links (rechts).
- ▶ Interessanter ist die allgemeinere Suche nach dem Knoten mit dem k -kleinsten Schlüssel.
 - Falls die Anzahl der Knoten im linken Teilbaum mindestens k ist:
 - Sind dort k Knoten, ist das größte Element dieses linken Teilbaums das k -kleinste Element,
 - sind dort mehr als k Knoten enthalten, wird die Suche rekursiv im linken Teilbaum fortgesetzt.
 - Andernfalls enthält der linke Teilbaum t Elemente mit $t < k$:
 - Ist $(k-t-1)$ null so ist man bereits auf dem k -kleinsten Knoten,
 - ansonsten wird das k -kleinste Element rekursiv als das $(k-t-1)$ -kleinste im rechten Teilbaum weiter gesucht.
- ▶ Für eine Implementierung dieses Verfahrens kann es nützlich sein, wenn jeder Knoten den Wert für die Größe seines Teilbaums bereits mitführt,
 - z.B. als Attribut `sz` (vgl. Code-Hinweis)...

```
int count( node* n ) {
    if( n==nullptr ) return 0;
    n->sz = count( n->left ) + count( n->right ) + 1;
    return n->sz;
}
```

Ausgeglichener binärer Suchbaum

Ausgeglichenheit, Vollständigkeit.

- ▶ Wünschenswert ist, dass Instanzen des Suchbaums möglichst *ausgeglichen* sind.
- ▶ D.h. dass auf jeder Ebene eines BST mit N Knoten alle Teilbäume ungefähr gleich viele Knoten beinhalten.
 - Genauer: auf jeder Ebene m sind 2^m Knoten, mit Ausnahme der höchsten Ebene, auf der maximal 2^m Knoten sind,
 - bzw. Höhe $h = \lfloor \log_2(N) \rfloor$.
- ▶ Je ausgeglichener, desto besser ist der ungünstigste Fall beim Suchen eines Elements.
- ▶ Sind alle Ebenen des BST außer der höchsten vollständig mit inneren Knoten besetzt, und die höchste Ebene *von links* mit inneren Knoten ausgestattet (d.h. innere Knoten "fehlen" auf der höchsten Ebene nur von rechts), spricht man auch von einem *vollständigen* BST.

Ausgeglichener binärer Suchbaum

Vertiefung.

- ▶ Ausgeglichene Suchbäume sind z.B. die sog. *Top-Down 2-3-4 Bäume*.
 - Diese Suchbäume sind nicht mehr binär und sind ein Spezialfall der sog. B-Bäume,
 - bzw. *Rot-Schwarz-Bäume*, die eine binäre Version der Top-Down 2-3-4-Bäume und äquivalent zu diesen sind.
- ▶ Die allgemeine Idee / Theorie für Top-Down 2-3-4 Bäume lässt sich leicht beschreiben,
- ▶ die praktische Umsetzung ist aber wegen der vielen zu berücksichtigenden Spezialfälle und Symmetrien recht zeitraubend.

Vertiefung: Top-Down 2-3-4 Bäume

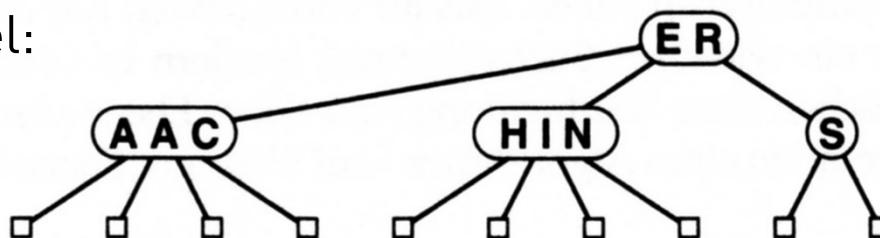
B-Baum der Ordnung m .

- ▶ Ein sog. B-Baum der Ordnung m – beschrieben von R. Bayer und E. McCreight (1972) – hat folgende Eigenschaften:
 - Alle externen Knoten befinden sich auf der selben Ebene (d.h.: perfekt ausgeglichener Baum),
 - jeder Knoten hat maximal m direkte Nachfolger,
 - jeder interne Knoten, mit Ausnahme der Wurzel, hat minimal $\lceil m/2 \rceil$ direkte Nachfolger,
 - die Wurzel hat minimal zwei direkte Nachfolger
 - (außer im leeren Baum, in dem die Wurzel bereits kein innerer Knoten mehr ist),
 - ein interner Knoten mit k Nachfolgern hat $k-1$ Schlüsselwerte.
- ▶ Top-Down 2-3-4 Bäume kann man als B-Bäume mit $m = 4$ verstehen.
- ▶ Wie wir gleich am Beispiel der Top-Down 2-3-4 Bäume sehen werden, bleiben solche Bäume immer perfekt ausgeglichen, wenn das Einfügen und Entfernen von Knoten entsprechend geschickt gestaltet wird...

Vertiefung: Top-Down 2-3-4 Bäume

Innere Knoten können zwei, drei oder vier Nachfolger haben (Spezialfall eines B-Baums der Ordnung 4).

- ▶ In diesen Suchbäumen gibt es drei Arten von inneren Knoten.
 - 2-Knoten, die einen Schlüsselwert und zwei Nachfolger haben:
 - Ein 2-Knoten definiert zwei Suchintervalle: links vom Suchschlüssel und rechts vom Suchschlüssel (die inneren Knoten im binären Suchbaum sind 2-Knoten).
 - 3-Knoten, die zwei Schlüsselwerte (einen kleinen und einen großen) und drei Nachfolger haben:
 - Ein 3-Knoten definiert drei Suchintervalle: links vom kleinen Suchschlüssel, zwischen den beiden Suchschlüsseln, rechts vom großen Suchschlüssel.
 - 4-Knoten, die drei Schlüsselwerte (einen kleinen, einen mittleren und einen großen) und vier Nachfolger haben:
 - Ein 4-Knoten definiert vier Suchintervalle: links vom kleinen Suchschlüssel, rechts vom kleinen und links vom mittleren Suchschlüssel, rechts vom mittleren und links vom großen Suchschlüssel, rechts vom großen Suchschlüssel.
- ▶ Beispiel:



Vertiefung: Top-Down 2-3-4 Bäume

Einfügen von Schlüsselwerten in Top-Down 2-3-4 Bäume.

- ▶ Um einen Wert einzufügen wird wie zuvor eine erfolglose Suche durchgeführt um dann an der Nullverbindung den Knoten anzuhängen.
- ▶ Falls der Knoten mit der Nullverbindung ein 2-Knoten ist, wird dieser in einen 3-Knoten umgewandelt und der neue Knoten als dritter Nachfolger eingefügt.
- ▶ Falls der Knoten mit der Nullverbindung ein 3-Knoten ist, wird dieser in einen 4-Knoten umgewandelt und der neue Knoten als vierter Nachfolger eingefügt.
- ▶ Der Fall, dass der Knoten, an dem der Wert eingefügt werden muss, ein 4-Knoten ist, wird verhindert,
 - indem bei jeder Einfügeoperation auf dem Weg von der Wurzel durch den Suchbaum nach unten zur Einfügestelle jeder angetroffene 4-Knoten, gewissermaßen vorsorglich, aufgespalten wird.
 - Der Begriff "Top-Down 2-3-4 Baum" kommt von diesem Vorgang.

Vertiefung: Top-Down 2-3-4 Bäume

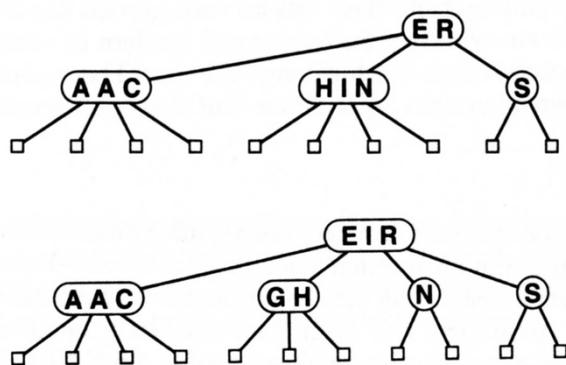
Einfügen von Schlüsselwerten in Top-Down 2-3-4 Bäume.

► Aufspaltung von 4-Knoten:

- Zuerst wird der 4-Knoten in zwei 2-Knoten zerlegt,
- dann wird einer der Schlüssel an den Vorgänger übergeben.



► Beispiel: G einfügen

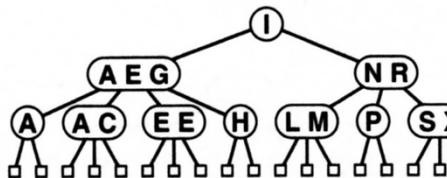
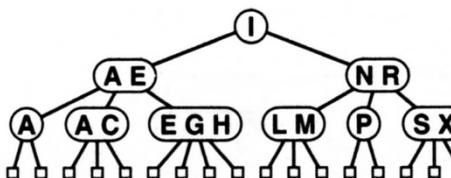
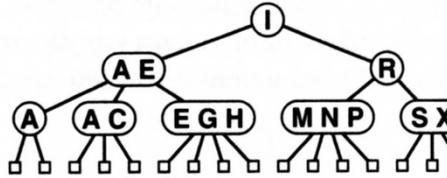
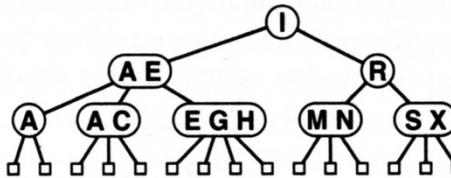
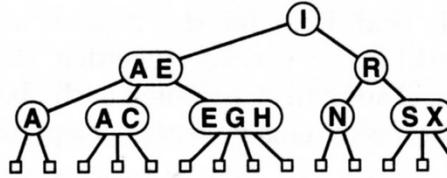
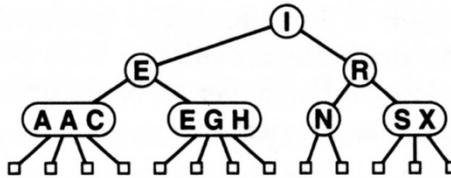
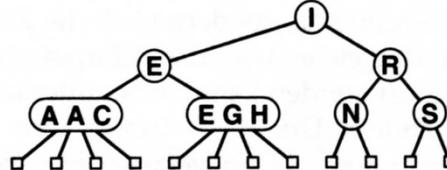
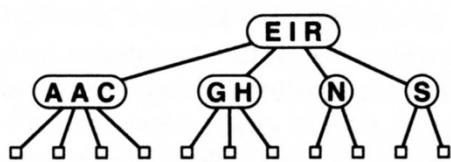


- Der 4-Knoten mit H, I und N wird in zwei 2-Knoten aufgespalten, wobei der linke H und der rechte N enthält.
- Der mittlere Schlüssel I wird nach oben an den 3-Knoten übergeben, der E und R enthält, wodurch dieser zu einem 4-Knoten wird.
- Danach ist im 2-Knoten, der H enthält, Platz für G.

Vertiefung: Top-Down 2-3-4 Bäume

Einfügen von Schlüsselwerten in Top-Down 2-3-4 Bäume.

- Weitere Einfügungen: E, X, A, M, P, L und E



Beobachtung:

Durch diese Methode des Einfügens bleiben Top-Down 2-3-4 Bäume immer perfekt ausgeglichen.

Vertiefung: Top-Down 2-3-4 Bäume

Durch die beschriebene Methode des Einfügens bleiben Top-Down 2-3-4 Bäume immer perfekt ausgeglichen.

► Erläuterung und Beweisskizze:

- (1) Es befinden sich in einem gültigen (d.h. perfekt ausgeglichenen) Top-Down 2-3-4 Baum alle externen Knoten auf der selben Baumebene h und ein neuer Schlüsselwert wird immer auf Ebene $h - 1$ eingefügt.
- (2) Enthält der Knoten, in dem der Wert einzufügen ist, weniger als drei Schlüssel, so kann einfach eingefügt werden und alle Eigenschaften des Baums bleiben erhalten, insbes. auch die vollständige Ausgeglichenheit, da sich immer noch alle externen Knoten auf der selben Ebene h befinden.
- (3) Enthält der Knoten, in dem der Wert einzufügen ist, schon drei Schlüssel, so wird er aufgespalten und ein Schlüssel wird im Vorgängerknoten auf Ebene $h - 2$ eingefügt. Wieder bleiben alle Eigenschaften des Baums, einschließlich der vollständigen Ausgeglichenheit, erhalten, denn nach wie vor befinden sich alle externen Knoten auf derselben Ebene h .
- (4) Schritt (3) gilt falls nötig rekursiv durch den kompletten Baum. Entweder endet die Rekursion an einem Vorgängerknoten, der nicht gespalten werden muss, vgl. Schritt (2). Oder die Rekursion scheitert beim Aufspalten an der Wurzel, da die Wurzel keinen Vorgänger hat, an den ein Schlüsselwert übergeben werden kann, vgl. Schritt (5).
- (5) Muss die Wurzel aufgespalten werden, wird eine neue Wurzel erzeugt, an die dann ein Schlüssel aus der (alten) Wurzel übergeben werden kann und die (alte) Wurzel wird aufgespalten. Auch hierbei bleibt die vollständige Ausgeglichenheit des Baums erhalten, es befinden sich alle Knoten (außer der neuen Wurzel) nun aber auf einer um eins höher gewordenen Ebene (alle externen Knoten sind nun auf derselben Ebene $h + 1$).
- (6) D.h. dieser Baum wächst immer nur an der Wurzel.
- (7) Ob die Aufspaltung "Top-Down" passiert (also vor dem Einfügen auf dem Weg durch den Baum zur Einfügestelle), oder umgekehrt wie hier für die Erläuterung beschrieben, ist für die Erhaltung der vollständigen Ausgeglichenheit egal.

► Für das Entfernen von Knoten gibt es ein ähnliches Verfahren,

- mit einer ähnlichen Methode, aber kniffliger (weil einige Fallunterscheidungen),
- d.h. dieser Baum schrumpft auch immer nur an der Wurzel.

LEERE SEITE

Streuwert-Tabellen

Streuwert-Tabellen sind ein klassischer Gegenstand der Informatik.

- ▶ Bei Streuwert-Tabellen (engl. Hash-Tables von hashing, "zerhacken") geht es um eine Methode für die direkte Bezugnahme auf Datensätze einer Tabelle (der Streuwert-Tabelle), indem Schlüsselwerte in Tabellenindizes umgerechnet werden.
- ▶ Streuwert-Tabellen sind in der Informatik recht gut untersucht worden und haben weite Verbreitung gefunden.
- ▶ Man kann grob davon ausgehen, dass in Streuwert-Tabellen Suche und Einfügung, unabhängig von der Anzahl der Elemente, in konstanter Laufzeit unterstützt werden.
- ▶ Diese theoretische Optimalleistung macht Streuwert-Tabellen dennoch nicht zum Wundermittel.
 - Die Laufzeit hängt zwar nicht von der Anzahl der Elemente, aber u.a. von der Länge der Schlüssel ab, d.h. bei Anwendungen mit sehr langen Schlüsseln können Probleme auftauchen.
 - Zudem bieten Streuwert-Tabellen keine effizienten Implementierungen für das Sortieren.

Streuwert-Tabellen

Streuwert-Tabellen beinhalten zwei grundlegende Verfahren: *erstens* eine Streuwertfunktion und *zweitens* ein Verfahren zur Kollisionsbeseitigung.

- ▶ Im *ersten Schritt* wird aus dem Suchschlüssel mittels einer *Streuwertfunktion* ein Tabellenindex erzeugt.
 - Da es normalerweise viel mehr theoretisch mögliche Schlüsselwerte als die tatsächlich verwendeten gibt, macht es wenig Sinn, für jeden möglichen Schlüsselwert Tabellenindizes vorzuhalten, da der entsprechende Speicherplatz weitgehend leer bleiben wird.
 - Die Streuwertfunktion bildet also viele Suchschlüssel auf wenige ganzzahlige sog. *Streuwerte* ab.
 - Im Idealfall so, dass unterschiedliche Schlüssel immer auch unterschiedliche Streuwerte (d.h. Tabellenindizes) erzeugen.
- ▶ Es kann aber passieren, dass zwei oder mehr unterschiedliche Schlüssel den gleichen Streuwert (also Tabellenindex) erzeugen.
 - Streuwertfunktionen bzw. ihre Erfinder können ja nicht "wissen", welche der möglichen Schlüsselwerte tatsächlich auftauchen...

Streuwert-Tabellen

Streuwert-Tabellen beinhalten zwei grundlegende Verfahren: erstens eine Streuwertfunktion und zweitens ein Verfahren zur Kollisionsbeseitigung.

- ▶ Der zweite Schritt führt eine *Kollisionsbeseitigung* durch.
 - Diese befasst sich mit dem Fall, dass für unterschiedliche Schlüssel durch die Streuwertfunktion gleiche ("kollidierende") Streuwerte erzeugt werden.
 - Das Verfahren muss mit allen diesen Fällen umgehen können,
 - immer, wenn die Streuwertfunktion aus einem anderen Input (Schlüsselwert) erneut einen "bereits vergebenen" Output (Streuwert/Tabellenindex) berechnet.
- ▶ Vgl. auch das sog. "Geburtstagsrätsel",
 - eine bekannte Knobelaufgabe: Wahrscheinlichkeit, dass von n zufällig ausgewählten Personen mindestens zwei am selben Tag Geburtstag haben?
 - Für die meisten, die es zum ersten Mal durchdenken, überraschend hoch:

$$p = 1 - \frac{365 \cdot 364 \cdot \dots \cdot (365-n+1)}{365^n} \quad (\text{ist schon ab } n = 23 \text{ größer als } \frac{1}{2})$$

Streuwert-Tabellen

Einfachster Fall: direkte Schlüsselindizierung mit ganzen Zahlen.

► Schlüsselindizierung:

- Es sollen Elemente mit eindeutigen Schlüsselwerten aus nicht zu großen Ganzzahlen gespeichert werden.
- Man kann dann die Elemente z.B. in einem entsprechend großen `vector` oder `array` speichern und *direkt* durch die Schlüssel k indizieren.
 - `data[k]` // etwas schneller
 - `data.at(k)` // sicherer (indexgeprueft)
- Es ist dann
 - weder eine eigentliche Streuwertfunktion
 - noch (für den Fall eindeutiger Schlüsselwerte) eine Kollisionsbehandlung nötig.

Streuwert-Tabellen

Einfachster Fall: direkte Schlüsselindizierung mit ganzen Zahlen.

► Beispiel:

- Die Schlüssel seien Ganzzahlen aus [0; 456975].
- Es sollen Wörter mit fester Länge (vier aus 26 Großbuchstaben) gespeichert werden.

```
using symbol4 = array< char, 4 >;      // feste Laenge 4 fuer jedes Wort
const size_t M { 456976U };                // 26^4
array< symbol4, M > sitab {};              // Platz fuer alle moeglichen Worte
setSym4( sitab.at(44217), "AKEY" );        // AKEY z.B. am Index 44217
setSym4( sitab[114156], "COOL" );          // COOL z.B. am Index 114156
//...
for( size_t k{}; k<M; ++k )
    if( sitab[k][0] ) printSym4( sitab[k] );
```

```
// die beiden Hilfsfunktionen:

void setSym4( symbol4& rs, const char* s ) {
    for( size_t i{}; i<4; ++i, ++s )
        rs[i] = *s;
}

void printSym4( symbol4& rs ) {
    cout << rs[0] << rs[1] << rs[2] << rs[3];
}
```

Streuwert-Tabellen

Streuwertfunktionen berechnen aus Schlüsselwerten die Tabellenindizes.

- ▶ Normalerweise werden viel weniger Schlüssel tatsächlich verwendet als es theoretisch mögliche Schlüssel gibt.
 - Im eben besprochenen Beispiel: 26^4 Schlüssel sind theoretisch möglich (alle Symbol-Kombinationen mit vier aus 26 Großbuchstaben).
 - Es macht wenig Sinn, wie in diesem Beispiel für alle theoretisch möglichen Schlüssel auch tatsächlich 456'976 Tabellenindizes / Speicherstellen vorzuhalten, wenn tatsächlich nur wenige (z.B. nur einige Hundert) der möglichen Symbole wirklich vorkommen können.
- ▶ Es könnte daher ein Rechenverfahren eingesetzt werden, das jeden der N theoretisch möglichen Schlüssel auf eine Ganzzahl im kleinen Bereich $[0; M-1]$ abbildet, dann würde eine direkt indexierbare Speicherstruktur mit einer Größe von nur M Elementen ausreichen.
 - Eine solche Speicherstruktur wird Streuwert-Tabelle genannt.
 - Ein solches Rechenverfahren wird Streuwertfunktion genannt.
 - Der Output, den die Streuwertfunktion für einen bestimmten Schlüsselwert berechnet, wird Streuwert dieses Schlüsselwerts genannt und ist der Index, an dem der Schlüsselwert in der Streuwert-Tabelle abgelegt wird.

Streuwert-Tabellen

Streuwertfunktionen berechnen aus Schlüsselwerten die Tabellenindizes.

- ▶ Die Definition einer Streuwertfunktion ist offensichtlich abhängig
 - vom Typ des Schlüssels, speziell der Möglichkeit zur Umrechnung von Werten dieses Typs in Ganzzahlen, die dann als Indizes verwendet werden,
 - und von der Möglichkeit, den Streuwert möglichst schnell aus dem Schlüsselwert zu berechnen.
- ▶ Oft lässt sich eine Streuwertfunktion recht einfach implementieren.
 - Besonders nahe liegend kann es etwa sein, die Binärdarstellung des Schlüssels zur Berechnung des Streuwalts zu verwenden.
 - Dies hat jedoch auch Nachteile, z.B.:
 - widerspricht es dem Konzept der Typsicherheit moderner Programmiersprachen
 - und es ist schwer, dies portabel und gleichzeitig effizient zu implementieren.
- ▶ Darüber hinaus sollen möglichst wenig kollidierende Streuwerte erzeugt werden.
 - Theoretisch ideal im Sinne dieser Anforderung wären über $[0; M-1]$ gleichverteilte Streuwerte, egal welche der Schlüsselwerte aus der großen Menge der theoretisch möglichen wirklich erscheinen,
 - damit sich die minimale Kollisionswahrscheinlichkeit von $1/M$ für zwei beliebige Schlüsselwerte ergibt.
 - Solche Rechenverfahren ("perfekte Streuwertfunktionen") sind schwer zu finden.

Streuwert-Tabellen

Gleitkommazahlen als Schlüsselwerte, *multiplikative Streuwertfunktionen*.

- ▶ Gleitkommazahlen aus $[0;1[$ als Schlüsselwerte können durch Multiplikation mit M und Abrundung zur nächsten Ganzzahl auf einen Index aus $[0; M-1]$ abgebildet werden.
 - Allgemeine Gleitkommazahlen aus $[s, t[$ als Schlüsselwerte können zuerst in das Intervall $[0;1[$ skaliert werden.
- ▶ Beispiel: die Streuwert-Tabellengröße sei $M = 97$.

0, 513870656	50
0, 175725579	17
0, 308633685	30
0, 534531713	52
0, 947630227	92
0, 171727657	17
0, 702230930	68
0, 226416826	22
0, 494766086	48
0, 124698631	12
0, 083895385	08
0, 389629811	38
0, 277230144	27
0, 368053228	36
0, 983458996	95
0, 535386205	52
0, 765678883	74
0, 646473587	63
0, 767143786	74
0, 780236185	76
0, 822962105	80
0, 151921138	15
0, 625476837	61
0, 314676344	31
0, 346903890	34

- In diesen 25 zufälligen Beispielschlüsseln aus $[0;1[$ treten *drei Kollisionen* auf: 17, 52 und 74.

Streuwert-Tabellen

Ganzzahlen als Schlüsselwerte, *modulare Streuwertfunktionen.*

- ▶ Ist der Schlüssel k eine Ganzzahl, verwendet man als Streuwertfunktion meist $h(k) = k \bmod M$.
 - Die sog. modulo-Operation, d.h. der Streuwert ist der Rest der ganzzahligen Division von k durch M .
- ▶ Beispiel:
 - Die Größe der Streuwert-Tabelle sei $M = 101$.
 - Ein Schlüsselwert k_1 sei 44217.
 - $44217 \equiv 80 \pmod{101}$
 - D.h. der Wert 44217 wird an Position 80 der Streuwert-Tabelle gespeichert.
 - Ein anderer Schlüsselwert k_2 sei 114156.
 - $114156 \equiv 26 \pmod{101}$
 - D.h. der Wert 114156 wird an Position 26 der Streuwert-Tabelle gespeichert.

Streuwert-Tabellen

Ganzzahlen als Schlüsselwerte, *modulare Streuwertfunktionen*.

- ▶ Kollisionen der modularen Streuwertfunktion können seltener gemacht werden, indem man die Größe M der Tabelle (= das Modul der Streuung) sorgfältig wählt.
- ▶ Man muss z.B. an folgende Punkte denken:
 - Ist M gerade, so ist $k \bmod M$ für gerade k immer gerade und für ungerade k immer ungerade.
 - Ist M die Potenz einer Zahl, so werden ggf. nicht alle Stellen des Schlüsselwerts zur Streuwertberechnung berücksichtigt.
 - Ist z.B. $M = 10^j$ werden durch $k \bmod M$ nur die rechten j Dezimalziffern von k verwendet, bei z.B. $M = 2^j$ gibt $k \bmod M$ einfach die rechten j Bits (die am wenigsten signifikanten) der Binärdarstellung von k zurück
 - Wird ein Vielfaches von M zum Schlüsselwert addiert, wird wieder derselbe Streuwert erzeugt, M sollte also vor allem nicht durch die Größe des verwendeten Zeichensatzes teilbar sein.
- ▶ Man wählt üblicherweise M prim, und nicht in der Nähe einer Potenz (speziell Zweierpotenz).

Streuwert-Tabellen

Kurze Zeichenketten als Schlüsselwerte für modulare Streuwertfunktionen.

- ▶ Kurze Zeichenketten können sehr einfach in Ganzzahlen umgerechnet werden, die dann modular gestreut werden können.
- ▶ Zurück zum Beispiel aus der direkten Schlüsselindizierung:
 - Die Schlüssel sind Zeichenketten aus vier von 26 Großbuchstaben,
 - wir könnten z.B. (wie vorher erwähnt, mit allen Vor- und Nachteilen) die 32-Bit Binärdarstellung der vier `char` direkt als `int` auffassen,
 - oder etwas portabler vorgehen und für den i -ten Buchstaben des Alphabets die binäre Kodierung der Zahl i verwenden:

Schlüssel	A	K	E	Y	C	O	O	L
i	1	11	5	25	3	15	15	12
i binär in 5 Bit	00001	01011	00101	11001	00011	01111	01111	01100
als 20 Bit int	44217				114156			

- Die vier 5-Bit-Werte können auch als Polynom geschrieben werden:

$$1 \cdot 32^3 + 11 \cdot 32^2 + 5 \cdot 32^1 + 25 \cdot 32^0 = 44'217$$

Bem.: $2^5 = 32$

$$3 \cdot 32^3 + 15 \cdot 32^2 + 15 \cdot 32^1 + 12 \cdot 32^0 = 114'156$$

- ▶ Nun kann man die ganzzahligen Werte in der vorigen modularen Streuwertfunktion, Streuwert-Tabelle der Größe $M = 101$, verwenden.

Streuwert-Tabellen

Lange Zeichenketten als Schlüsselwerte für modulare Streuwertfunktionen.

- ▶ Rechnet man lange Zeichenketten wie beschrieben um, dann werden die resultierenden Zahlen für die normalen arithmetischen Operationen der meisten Computer zu groß.
- ▶ Modulare Streuwertfunktionen für lange Zeichenketten nutzen daher einen Rechentrick, das sog. Hornerschema, in Verbindung mit der modulo-Operation.
 - Hornerschema:
$$44271 = 1 \cdot 32^3 + 11 \cdot 32^2 + 5 \cdot 32^1 + 25 \cdot 32^0 = ((1 \cdot 32 + 11) \cdot 32 + 5) \cdot 32 + 25$$
$$114156 = 3 \cdot 32^3 + 15 \cdot 32^2 + 15 \cdot 32^1 + 12 \cdot 32^0 = ((3 \cdot 32 + 15) \cdot 32 + 15) \cdot 32 + 12$$
 - modulo-Bildung:

Bei jeder Multiplikation/Addition im Hornerschema wird nur mit dem Rest modulo M weiter gerechnet: $((1 \cdot 32 + 11 \bmod M) \cdot 32 + 5 \bmod M) \cdot 32 + 25 \bmod M$
 - *Damit erhält man den Streuwert, ohne den Schlüssel komplett zur Ganzzahl umzurechnen.*

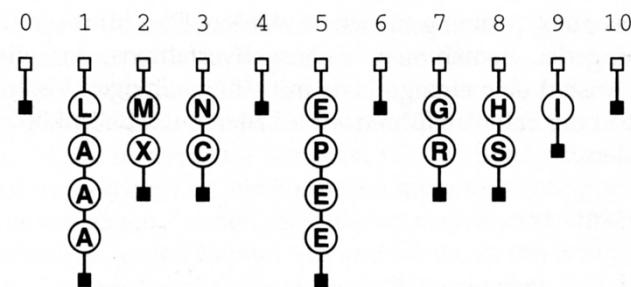
```
// so kann ein modularer Streuwert fuer lange
// Zeichenketten wie beschrieben berechnet werden
int hash( string s, int M ) {
    int h{}, a{31};
    for( unsigned int i{}; i < s.size(); ++i )
        h = (a*h + s[i]) % M;
    return h;
}
```

Streuwert-Tabellen

Eine Methode der Kollisionsbehandlung ist die "getrennte Verkettung".

- ▶ Bei der sog. *getrennten Verkettung* behandelt man kollidierende Streuwerte dadurch, dass man für jeden Tabellenindex eine verkettete Liste erzeugt, die alle Knoten enthält, deren Schlüssel auf diesen Index abgebildet wurden.
- ▶ Zur Implementierung der Streuwert-Tabelle kann man M verkettete Listen anlegen (eine pro Tabellenindex), in denen sequentiell gesucht wird, nachdem zuerst die Streuwertfunktion zur Auswahl der richtigen Liste verwendet wurde.
- ▶ Symbolisches Beispiel:
 - Streuwertfunktion, $M = 11$
 - Streuwert-Tabelle mit getrennter Verkettung

Schlüssel: A S E A R C H I N G E X A M P L E
Hash: 1 8 5 1 7 3 8 9 3 7 5 2 1 2 5 1 5



Streuwert-Tabellen

Eine weitere Methode der Kollisionsbehandlung ist die "offene Adressierung".

- ▶ Streuwertmethoden mit sog. *offener Adressierung* können unter zwei Voraussetzungen eingesetzt werden:
 - Die maximale Anzahl N^{max} der Elemente, die gespeichert werden sollen, kann vorher ungefähr abgeschätzt werden.
 - Man hat genug zusammenhängenden Speicher.
- ▶ Unter diesen Voraussetzungen kann man, statt verketteter Listen o.ä. in der Streuwert-Tabelle zu verwalten,
 - die Streuwert-Tabelle gleich als Datenfeld der Größe $M > N^{max}$ erzeugen
 - und bei Kollisionen auf die freien Plätze in der Streuwert-Tabelle ausweichen.
- ▶ Eine einfache Implementierung der offenen Adressierung ist das sog. *lineare Austesten*:
 - Wenn eine Kollision in der Streuwert-Tabelle vorliegt (d.h. wenn die Streuwertfunktion einen Index liefert hat, an dem schon ein Element mit einem Schlüssel ungleich dem Suchschlüssel gespeichert ist), dann wird der nächsthöhere Tabellenindex gegen den Suchschlüssel getestet.
 - Dies wird so lange durch die *ganze* Streuwert-Tabelle fortgesetzt, bis entweder der Suchschlüssel oder ein leerer Tabellenplatz gefunden wird.

Streuwert-Tabellen

Streuwert-Tabellen sind ein Beispiel für eine suchoptimierte Datenstruktur, die einen beliebigen Kompromiss zwischen Zeit- und Platzbedarf realisiert.

- ▶ Ohne Speicherbeschränkung, mit beliebig großen Streuwert-Tabellen, könnte jede Suche mit nur einem einzigen Speicherzugriff realisiert werden
 - indem man einfach den Schlüsselwert direkt als Speicheradresse interpretiert,
 - vgl. direkte Schlüsselindizierung.
- ▶ Ohne Zeitbeschränkung könnte jede Suche mit nur minimalem Speicherplatz realisiert werden
 - indem ein sequentielles Suchverfahren eingesetzt wird,
 - vgl. verkettete Listen.
- ▶ Mit Streuwert-Tabellen kann man jedes beliebige Verhältnis zwischen diesen beiden Extrema einstellen,
 - *nur* durch Anpassung der Größe der Streuwert-Tabelle,
 - *ohne* neuen Quellcode oder andere Algorithmen.

Einige Beispielfragen

Fortgeschrittenes Suchen.

- ▶ Welchen Wert haben die harmonischen Zahlen H_1, H_2, H_4 ?
- ▶ Woraus besteht ein Baum?
- ▶ Welche entscheidende Eigenschaft zeichnet die Bäume unter den Graphen aus?
- ▶ Was ist der Unterschied zwischen einem Wurzelbaum und einem freien Baum?
- ▶ Was ist ein Blatt?
- ▶ Was verstehen Sie unter einer Knotenebene?
- ▶ Was verstehe Sie unter der Höhe eines Wurzelbaums?
- ▶ Wann kann man von einem geordneten Baum sprechen?
- ▶ Was ist das Besondere an einem M -wertigen Baum?
- ▶ Warum gibt es in M -wertigen Bäumen zwei Arten von Knoten? Wie nennt man diese?
- ▶ Wie viele Nachfolger hat ein Blatt eines M -wertigen Baums?

Einige Beispielfragen

Fortgeschrittenes Suchen.

- ▶ Erklären Sie, wie ein Binärbaum geordnet ist.
- ▶ Was verstehen Sie unter der Traversierung eines Binärbaums?
- ▶ Welche Methoden zur Traversierung eines Binärbaums kennen Sie? Erklären Sie jeweils, wie diese funktionieren.
- ▶ Welchen Unterschied sehen Sie zwischen einem Binärbaum und einem binären Suchbaum?
- ▶ Geben Sie ein *eigenes* Beispiel für einen gültigen Binärbaum, der kein binärer Suchbaum ist.
- ▶ Welchen Zweck haben binäre Suchbäume?
- ▶ Erklären Sie die Ordnungssystematik in einem binären Suchbaum mit eigenen Worten, ohne Formeln, ohne Quellcode-Zitate.
- ▶ Skizzieren Sie den Ablauf einer Suchoperation in einem binären Suchbaum.
- ▶ Wo würden Sie einen neuen Knoten in einen binären Suchbaum einfügen? Wie würden Sie vorgehen, um die Einfügungsstelle zu finden?

Einige Beispielfragen

Fortgeschrittenes Suchen.

- ▶ Wie geben Sie alle Werte in einem binären Suchbaum aufsteigend sortiert aus?
- ▶ Warum kann es vorteilhaft sein, wenn neue Knoten in binären Suchbäumen "als Wurzel eingefügt" werden?
- ▶ Erläutern Sie das in der Vorlesung vorgestellte Verfahren, mit dem man einen neuen Knoten in einen binären Suchbaum "als Wurzel einfügt". Warum fügt man den Knoten nicht einfach ganz oben ein und hängt danach links bzw. rechts die "alte" Wurzel an?
- ▶ Welche Fälle werden unterschieden, wenn man einen Knoten aus einem binären Suchbaum entfernen will?
- ▶ Wann bezeichnet man Suchbäume als ausgeglichen?
- ▶ Nennen Sie einen Suchbaum, der seine eigene Ausgeglichenheit garantiert.

Einige Beispielfragen

Fortgeschrittenes Suchen.

- ▶ Was verstehen Sie unter einer Streuwert-Tabelle? Wie lautet der englische Begriff?
- ▶ Welche beiden grundsätzlichen Verfahren sind für Streuwert-Tabellen zu beachten?
- ▶ Was ist ein Streuwert? Was ist eine Streuwertfunktion?
- ▶ Was sind schlüsselindizierte Datenfelder? Wann kann man sie einsetzen?
- ▶ Warum sind Streuwert-Tabellen i.Allg. viel kleiner als die Anzahl der theoretisch möglichen Schlüssel, die auftreten könnten?
- ▶ Erklären Sie mit eigenen Worten, wie Schlüsselwerte, Streuwertfunktionen, Streuwerte und Streuwert-Tabellen zusammenhängen.
- ▶ Welche Vorteile hat es, die Binärdarstellung eines Schlüsselwerts direkt als seinen Streuwert zu interpretieren? Welche Nachteile hat es?
- ▶ Was verstehen Sie unter einer multiplikativen Streuwertfunktion?
- ▶ Was verstehen Sie unter einer modularen Streuwertfunktion?

Einige Beispielfragen

Fortgeschrittenes Suchen.

- ▶ Wie kann man modulare Streuwertfunktionen für kurze Zeichenketten-Schlüssel einsetzen?
- ▶ Welche Schwierigkeit ergibt sich, wenn man modulare Streuwertfunktionen für lange Zeichenketten-Schlüssel einsetzt?
- ▶ Erklären Sie, wie man Streuwerte aus langen Zeichenketten berechnet, ohne dabei den Schlüsselwert vollständig auszurechnen.
- ▶ Warum braucht man eine Kollisionsbehandlung in Streuwert-Tabellen?
- ▶ Wie löst eine Streuwert-Tabelle mit getrennter Verkettung Streuwert-Kollisionen auf?
- ▶ Wie löst eine Streuwert-Tabelle mit offener Addressierung Streuwert-Kollisionen auf? Wie wird offene Adressierung als lineares Austesten implementiert?
- ▶ Wie könnte man ganz allgemein die durchschnittlichen Antwortzeiten einer Streuwert-Tabelle verbessern?

Nächste Einheit:

Fortgeschrittenes Sortieren