

j-Algo

The Algorithm
Visualisation Tool

Benutzerhandbuch

Inhaltsverzeichnis

I. Das Hauptprogramm	5
1. Das Hauptprogramm	7
1.1. Einleitung	7
1.2. Technische Hinweise	7
1.2.1. Systemvoraussetzungen	7
1.2.2. Installation	7
1.2.3. Deinstallation	8
1.3. Grundfunktionen	9
1.3.1. Neues Modul öffnen	9
1.3.2. Gespeicherte Sitzungsdaten laden	9
1.3.3. Sitzungsdaten speichern	10
1.3.4. Modul schließen	10
1.3.5. Einstellungen	10
1.3.6. Hilfe	11
1.3.7. Hinweis-Tipps	11
1.4. Impressum	12
II. Die Module	13
2. Das Modul AVL-Bäume	15
2.1. Einleitung	15
2.2. Funktionsübersicht	15
2.3. Programmstart - Der Willkommensbildschirm	16
2.3.1. Baum laden	16
2.3.2. Baum von Hand erstellen	16
2.3.3. Zufallsbaum erstellen lassen	17
2.3.4. Willkommensbildschirm anzeigen	17
2.4. Die Arbeitsfläche	18
2.5. Modulfunktionen	19
2.5.1. Schlüsseingabe	19
2.5.2. Algorithmusfunktionen	20
2.5.3. AVL-Modus	20
2.5.4. Baum auf AVL-Eigenschaft testen	20
2.5.5. Baum löschen	21
2.6. Algorithmussteuerung	21

Inhaltsverzeichnis

2.6.1.	Schritt-Pfeile	21
2.6.2.	Abbruch und Beenden-Buttons	22
2.6.3.	Animationsgeschwindigkeit	22
2.7.	Dokumentation	22
2.7.1.	Skript	22
2.7.2.	Logbuch	23
2.7.3.	Infobereich	23
2.8.	Zusatzfunktionen	23
2.8.1.	Navigator	23
2.8.2.	Beamernmodus	24
2.9.	Impressum	25
3.	Das Modul Dijkstra	27
3.1.	Einleitung	27
3.2.	Funktionsübersicht	27
3.3.	Modul starten	27
3.4.	Symbolleiste	27
3.5.	Graph erstellen	28
3.5.1.	Graphische Eingabe/Erstellen eines Graphen per Maus	28
3.5.2.	Die Knotenliste	29
3.5.3.	Die Kantenliste	29
3.5.4.	Die Adjazenzmatrix	29
3.6.	Ablauf des Algorithmus	30
3.7.	Impressum	31
A.	Einleitung zu Datenstrukturen	33
B.	Suchbäume	35
C.	AVL-Bäume	39

Teil I.

Das Hauptprogramm

1. Das Hauptprogramm

1.1. Einleitung

Dieses Handbuch stellt eine Einführung und Hilfe für die Arbeit mit **j-Algo** dar. **j-Algo** ist eine Software, die sich mit der Visualisierung von Algorithmen beschäftigt. Sie soll dazu dienen, verschiedene Algorithmen zu veranschaulichen um sie so Studenten und anderen Interessierten verständlicher zu machen. Die Anwendung basiert auf einer Plugin-Struktur, die es ermöglicht, einzelne Module, die jeweils einen Algorithmus oder ein Themengebiet abdecken können, in das Programm zu integrieren und zu laden.

Sowohl **j-Algo** als auch die einzelnen Module entstanden im Rahmen des externen Softwarepraktikums im Studiengang Informatik der TU Dresden in Zusammenarbeit mit dem Lehrstuhl Programmierung. Die implementierten Module orientieren sich daher an den Lehrveranstaltungen „Algorithmen und Datenstrukturen“ sowie „Programmierung“ im Grundstudium Informatik an der TU Dresden. Das Einsatzgebiet soll vor allem die Vorlesung und das studentische Lernen zu Hause umfassen.

j-Algo ist eine freie Software, die beliebig oft kopiert werden darf.

1.2. Technische Hinweise

1.2.1. Systemvoraussetzungen

Folgende minimale Systemanforderungen werden für den reibungslosen Einsatz von **j-Algo** benötigt:

- IBM-kompatibler PC
- Mindestens 64 MB RAM
- WINDOWS 98(SE)/ME/2000/XP , LINUX SuSE/Red Hat
- Java 2 Platform Standard Edition 5.0 (siehe: <http://java.sun.com/>)
- Maus und Tastatur
- Monitor mit einer Auflösung von mindestens 800x600

1.2.2. Installation

Windows

Entpacken Sie nach dem Herunterladen das ZIP-komprimierte Archiv in einen Ordner Ihrer Wahl. In diesem Ordner finden Sie eine Datei namens „j-algo.bat“. Öffnen Sie diese Datei mit einem Doppelklick, und das Programm wird gestartet.

1. Das Hauptprogramm

Unix

Entpacken Sie nach dem Herunterladen das TGZ-komprimierte Archiv in einen Ordner Ihrer Wahl. In diesem Ordner finden Sie eine Datei namens „j-algo.sh“. Öffnen Sie die Konsole und starten sie mittels `sh j-algo.sh` das Programm.

1.2.3. Deinstallation

Der komplette Programmordner kann jederzeit gefahrlos von der Festplatte gelöscht werden.

1.3. Grundfunktionen

j-Algo bietet eine Reihe von Grundfunktionen, die unabhängig von den Modulen zur Verfügung stehen, bzw. für jedes Modul die gleiche Bedeutung haben. Im Einzelnen sind das das Öffnen von Modulen sowie das Laden und Speichern von Sitzungsdaten. Die Grundfunktionen sind über die Werkzeugleiste oder den Menüpunkt <DATEI> erreichbar.

1.3.1. Neues Modul öffnen

Ein Klick auf den Button <NEU> in der Werkzeugleiste gibt Ihnen die Möglichkeit, ein beliebiges neues Modul zu öffnen. Dabei wird ein Auswahldialog geöffnet, in welchem die installierten Module aufgelistet sind. Hier werden Ihnen außerdem kurze Informationen zu diesen Modulen angezeigt. Sie können wählen, ob dieser Auswahldialog bei jedem Start des Programmes angezeigt werden soll oder nicht.

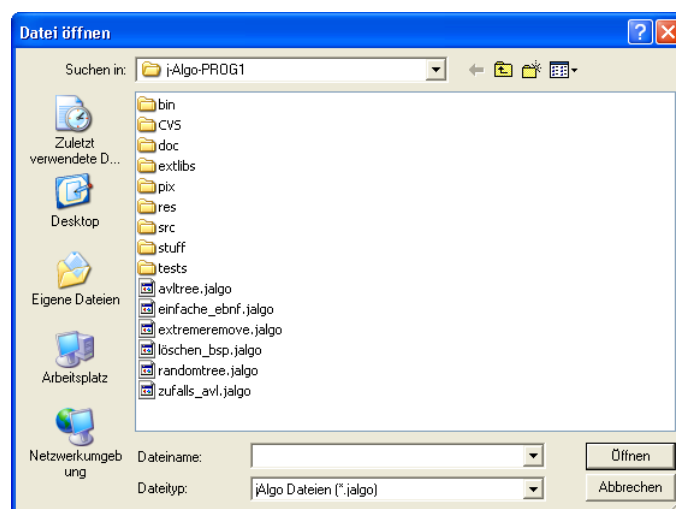
Alternativ dazu kann über das Menü <DATEI>→<NEU> das gewünschte Modul geladen werden. Dies ist der schnellere Weg und zu empfehlen, wenn man bereits einen Überblick über die installierten Module hat.

1.3.2. Gespeicherte Sitzungsdaten laden

Mit einem Klick auf den Button <ÖFFNEN> erscheint ein Dialog zur Dateiauswahl. Hier haben Sie die Möglichkeit, eine Datei auszuwählen, in welcher modulspezifische Sitzungsdaten gespeichert wurden. Die Dateien, die von **j-Algo** gespeichert werden, tragen die Dateiendung „.jalgo“.

Achtung: Da jedes Modul von **j-Algo** seine Daten in einer solchen Datei ablegt, kann man beim Blick auf die ungeöffnete Datei nicht erkennen, mit welchem Modul diese assoziiert wurde. Es wird jeweils das assoziierte Modul zu der geladenen Datei geöffnet. Achten Sie daher bei der Vergabe der Dateinamen auf möglichst eindeutige Bezeichner.

Anmerkung: In einer späteren Version wird direkt bei der Dateiauswahl das zugehörige Modul mit angezeigt.



Das Dialogfenster zum Öffnen

1. Das Hauptprogramm

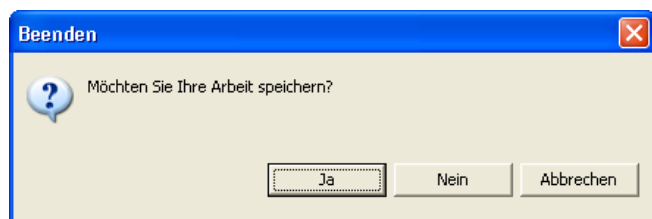
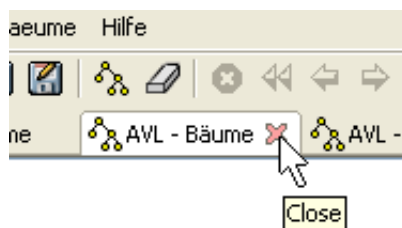
1.3.3. Sitzungsdaten speichern

Per Klick auf die Buttons <SPEICHERN> und <SPEICHERN UNTER> können Sie die Sitzungsdaten des gerade aktiven Moduls in einer Datei speichern. Wie beim Laden öffnet sich auch hier ein Dialog zur Dateiauswahl, in welchem Sie Zielpfad und Name der neuen Datei eintragen können. Die Angabe der Dateiendung ist nicht nötig, das Programm ergänzt diese automatisch.

Je nach Implementierung des aktiven Moduls steht die Speicherfunktion nur zur Verfügung, wenn gerade kein Algorithmus läuft. Sollte noch ein Algorithmus aktiv sein, so beenden Sie diesen bitte vorher oder brechen ihn ab.

1.3.4. Modul schließen

Sie haben die Möglichkeit, jede Modulinstanz durch Klick auf das Kreuz der dazugehörigen Registerkarte zu schließen. Dabei werden Sie gegebenenfalls gefragt, ob Sie Ihre Arbeit speichern wollen. Um das gesamte Programm zu schließen, ist es nicht nötig, die Module einzeln zu schliessen, das erledigt das Programm für Sie.

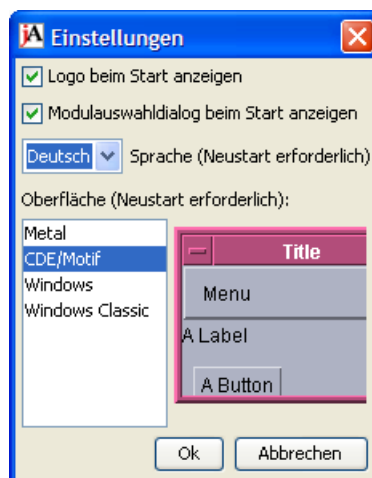


Der Knopf zum Schließen eines Moduls.

Die Abfrage, ob die Daten gespeichert werden sollen.

1.3.5. Einstellungen

j-Algo bietet ein paar Möglichkeiten an, das Programm den Bedürfnissen des Benutzers anzupassen. Den Dialog für die Grundeinstellungen erreichen Sie unter dem Menüpunkt <DATEI>→<EINSTELLUNGEN>



Der Dialog für die Grundeinstellungen

Unter anderem kann hier die Sprache eingestellt werden. Außerdem bietet sich die Möglichkeit, die Art der graphischen Oberfläche zu ändern, da unter manchen Betriebssystemen diverse Oberflächenelemente nicht immer vorteilhaft aussehen.

1.3.6. Hilfe

Die Hilfe stellt ein wichtiges Nachschlagewerk für all diejenigen dar, die nicht auf Anhieb mit allen Funktionen von **j-Algo** und seinen Modulen klar kommen. Hier können Sie noch einmal eine genaue Beschreibung zu den einzelnen Programmelementen nachlesen.

Die Hilfe ist kontextspezifisch aufgebaut, d.h. ist ein Modul geöffnet, so wird in der Hilfe automatisch an die entsprechende Stelle gesprungen.

Sie erreichen die Hilfe über den Menüpunkt `<HILFE>→<INHALT>` oder indem Sie einfach auf die Taste `<F1>` Ihrer Tastatur drücken.

1.3.7. Hinweis-Tipps

Zusätzlich wird zu den meisten Kontrollelementen, also Buttons, Menüeinträge, etc., ein kurzer Hinweistext neben dem Mauszeiger bzw. in der Statuszeile des Programmes angezeigt. Dies sollte als schnelle Hilfestellung den meisten Anforderungen genügen.

1. Das Hauptprogramm

1.4. Impressum

Die **j-Algo** Software wurde im Sommersemester 2004 von der Praktikumsgruppe SWT04-PROG1 im Rahmen des externen Softwarepraktikums entwickelt. Mitwirkende waren die

Teammitglieder

- Michael Pradel — Chief of Algorithms
- Cornelius Hald — Chief of Framework
- Malte Blumberg
- Stephan Creutz
- Christopher Friedrich
- Anne Kersten
- Hauke Menges
- Babett Schalitz
- Benjamin Scholz
- Marco Zimmerling

Die Webseite des Praktikums finden Sie unter <http://web.inf.tu-dresden.de/~swt04-p1/>. Komplette Überarbeitung erfuhr die Software und die Dokumentation unter anderem durch Alexander Claus und Matthias Schmidt. Weitergehende Informationen über **j-Algo** erhalten Sie unter <http://j-algo.binaervarianz.de/>.

Teil II.

Die Module

2. Das Modul AVL-Bäume

2.1. Einleitung

Das Modul **AVL-Bäume** realisiert die Darstellung von binären Such- und AVL-Bäumen. Für eine detaillierte Beschreibung dieser Baumtypen lesen Sie bitte nach im Vorlesungsskript von Prof. Vogler „Algorithmen, Datenstrukturen und Programmierung“. Eine mehr oder weniger kurze Einführung in diese Thematik ist auch zu finden im Anhang [A](#).

Anmerkung: In dieser Version von **j-Algo** ist das Modul **AVL-Bäume** unter Linux aus Kompatibilitätsgründen nicht installiert.

2.2. Funktionsübersicht

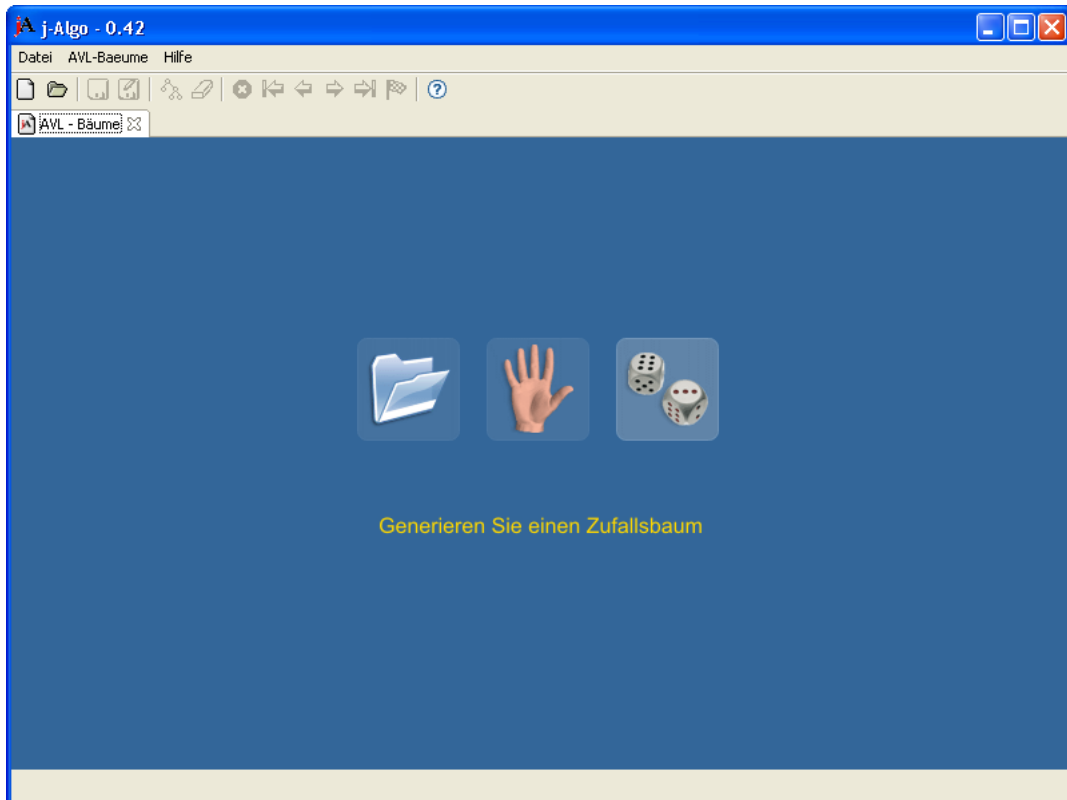
Das Modul **AVL-Bäume** realisiert folgende Funktionen:

- Visualisieren von binären Suchbäumen mit und ohne AVL-Eigenschaft
- Einfügen, Suchen und Löschen von Baumknoten
- Testen eines Baumes auf die AVL-Eigenschaft
- Generieren von zufälligen Suchbäumen
- Speichern und Laden von Bäumen
- Informationen zum Baum und zu den laufenden Algorithmen

2. Das Modul AVL-Bäume

2.3. Programmstart - Der Willkommensbildschirm

Nach Starten des Hauptprogramms **j-Algo** können Sie über den Button <NEU> oder mit dem Menüpunkt <DATEI>→<NEU>→<**AVL-BÄUME**> eine neue Instanz des Moduls **AVL-Bäume** öffnen. Anschließend öffnet sich der Willkommensbildschirm des Moduls, der Ihnen verschiedene Möglichkeiten eröffnet.



Der Willkommensbildschirm des Moduls **AVL-Bäume**

2.3.1. Baum laden

Mit Klick auf das Ordner-Symbol öffnet sich ein Dialogfenster, in dem Ihnen die Möglichkeit gegeben wird, eine „*.jalgo“-Datei auszuwählen, in welcher ein Baum gespeichert wurde. Im Prinzip ist die Bedeutung dieses Buttons die gleiche wie des <ÖFFNEN>-Buttons in der Werkzeugleiste. Der Unterschied besteht darin, dass der Button in der Werkzeugleiste eine neue Modulinstanz öffnet, in welcher die Datei geladen wird, der Button im Startbildschirm von **AVL-Bäume** jedoch die Datei in die aktuell geöffnete Modulinstanz lädt.

2.3.2. Baum von Hand erstellen

Mit Klick auf das Hand-Symbol gelangen Sie sofort zur leeren Arbeitsfläche des Moduls **AVL-Bäume**. Sie können jetzt mit der knotenweisen Generierung eines neuen Suchbaumes beginnen.

2.3.3. Zufallsbaum erstellen lassen

Mit Klick auf das Würfel-Symbol beginnen Sie die Generierung eines zufällig erzeugten Suchbaumes. In dem folgenden Dialogfenster können Sie verschiedene Daten zum Baum und die Art der Visualisierung festlegen.



Eingabe der Zufallsbaumdaten

- **Anzahl der Knoten**

Geben Sie hier die Anzahl der Knoten ein. Der entstehende Baum muss mindestens einen Knoten enthalten, höchstens aber 99.

- **AVL-Eigenschaft**

Aktivieren Sie dieses Kästchen, wenn der zu erstellende Baum die AVL-Eigenschaft besitzen soll.

- **Visualisierung**

Wählen Sie hier die Art der Visualisierung der Erstellung aus.

- KEINE

Der Baum wird sofort erstellt.

- SCHRITTWEISE

Jeder Algorithmusschritt kann von Ihnen per Hand bestätigt werden.

- AUTOMATISCH

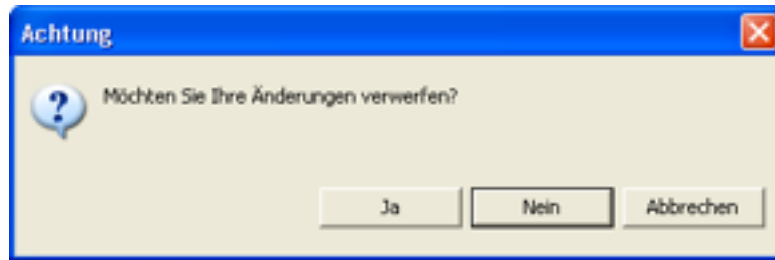
Lassen Sie die Erstellung des Baumes als Animation ablaufen, die Geschwindigkeit ist dabei einstellbar.

Haben Sie schrittweise oder automatische Visualisierung gewählt, können Sie den Ablauf jederzeit abbrechen. Dabei wird das gerade aktive Knoteneinfügen abgebrochen, und der Baum steht mit entsprechend weniger Knoten zur Verfügung.

2.3.4. Willkommensbildschirm anzeigen

Mit Klick auf diesen Button in der Werkzeugleiste des Modulbildschirms kann der Willkommensbildschirm später jederzeit wieder angezeigt werden. Dabei werden Sie eventuell gefragt, wie Sie mit Ihren Änderungen verfahren wollen. Sollten Sie Ihre Änderungen nicht verwerfen wollen, so wird eine neue Instanz des Moduls geöffnet.

2. Das Modul AVL-Bäume



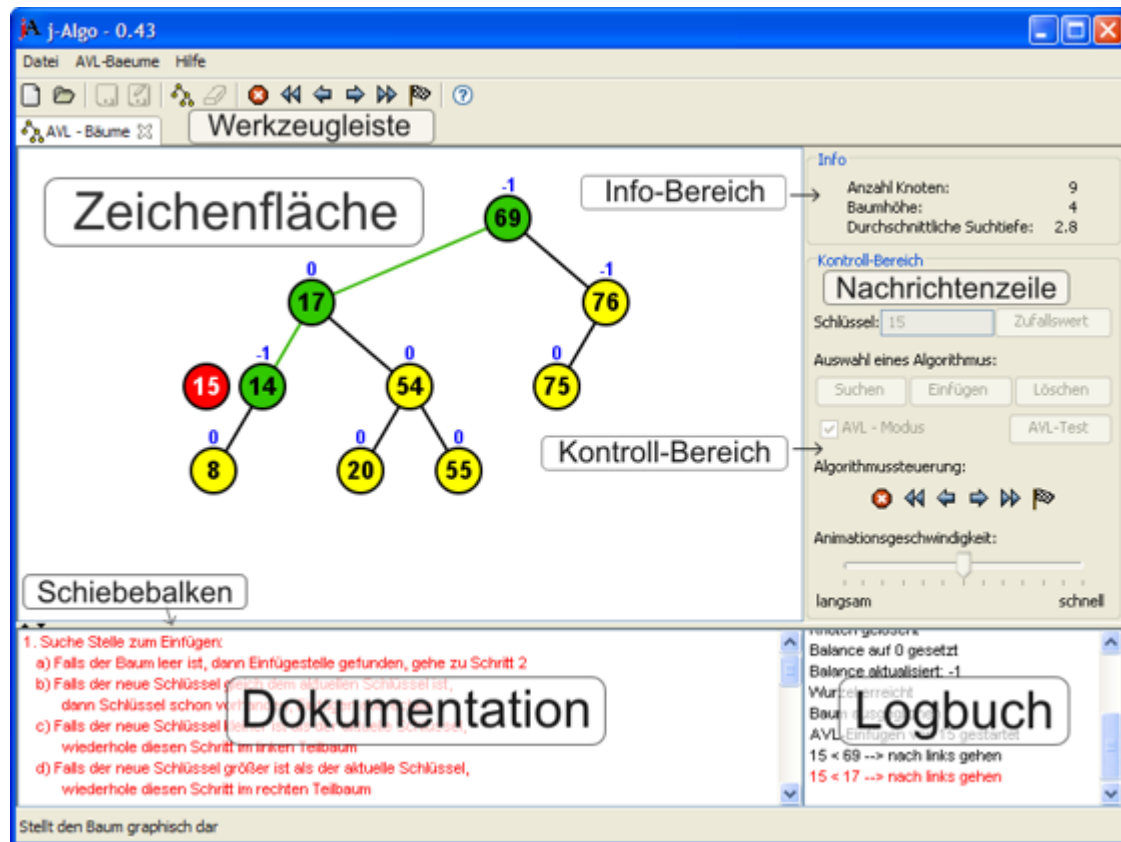
Dialog mit der Frage, ob der ganze Baum gelöscht werden soll.

2.4. Die Arbeitsfläche

Die Arbeit mit **AVL-Bäume** spielt sich auf der Arbeitsfläche ab. Sie bietet alle Funktionalitäten des Moduls und ist in fünf wichtige Bereiche unterteilt:

- **ZEICHENFLÄCHE**
Der Baum und alle Algorithmen werden hier visualisiert.
- **INFOBEREICH**
Wichtige Baumdaten wie Anzahl der Knoten, Baumhöhe und Suchtiefe sind hier zu finden.
- **KONTROLL-BEREICH**
In diesem Bereich erfolgt der Start und die Steuerung der Algorithmen.
- **DOKUMENTATIONSBEREICH**
Hier läuft der Text zum jeweiligen Algorithmus mit. Der aktuelle Schritt wird dabei farbig hervorgehoben. Der Text ist dem Skript „Algorithmen, Datenstrukturen und Programmierung“ von Prof. Vogler, Version vom 2. Oktober 2003, entnommen.
- **LOGBUCH**
Hier werden erfolgte Einzelaktionen protokolliert und dabei der jeweils aktuelle Schritt farbig hervorgehoben.

Die Aufteilung zwischen dem unteren und dem oberen Bereich kann mit dem Schiebebalken verändert werden. Per Klick auf die schwarzen Pfeile können Sie den Textbereich wahlweise maximieren, um einen Überblick über den Algorithmustext zu gewinnen, oder minimieren, um die Zeichenfläche zu vergrößern.

Die Arbeitsfläche des Moduls **AVL-Bäume**

2.5. Modulfunktionen

Alle Funktionen des Moduls **AVL-Bäume** lassen sich über den Kontroll-Bereich der Arbeitsfläche bedienen. Sie stellen die verschiedenen Baumalgorithmen dar, deren Visualisierung Aufgabe dieses Moduls ist. Grundslegend läuft die Arbeit mit den Algorithmen immer nach dem gleichen Schema ab:

1. Schlüsseleingabe
2. Starten des Algorithmus per Klick auf den entsprechenden Button

Es gibt natürlich auch Algorithmen, wie der AVL-Test, die keinen Schlüssel benötigen und ohne Schritt 1 auskommen.

Es folgen nun die einzelnen Funktionen im Detail.

2.5.1. Schlüsseleingabe

Für die Eingabe der Schlüsselwerte steht ein Textfeld und ein Button für zufällige Werte zur Verfügung. Es sind nur ganzzahlige Schlüsselwerte von 1 bis 99 erlaubt.

Über dem Textfeld befindet sich eine Nachrichtenzeile, in welcher Sie auf eventuelle Fehleingaben aufmerksam gemacht werden. Hier werden später ebenfalls kurze Ergebnismeldungen zu den Algorithmen eingeblendet.

2. Das Modul AVL-Bäume

2.5.2. Algorithmusfunktionen

Knoten einfügen

Der eingegebene Wert wird als Schlüssel für einen neuen Knoten verwendet, der in den Baum eingefügt werden soll. Ist bereits ein Knoten mit dem gleichen Schlüssel im Baum enthalten, so bricht der Algorithmus erfolglos ab.

Knoten suchen

Nach dem Starten dieses Algorithmus beginnt die Suche nach dem eingegebenen Schlüssel im Baum.

Knoten löschen

Nach dem eingegebenen Schlüssel wird gesucht, und wenn ein entsprechender Knoten gefunden wurde, wird dieser aus der Baumstruktur entfernt.

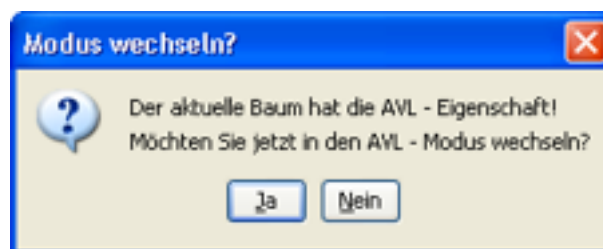
2.5.3. AVL-Modus

Ist dieses Kästchen aktiviert, werden die entsprechenden Algorithmen so ausgeführt, dass die AVL-Eigenschaft gewahrt bleibt.

Achtung: Es ist keine Funktion implementiert, die an einem beliebigen Suchbaum die AVL-Eigenschaft herstellt!

Ist das Kästchen deaktiviert, ist es daher nicht immer ohne weiteres wieder zu aktivieren. Dazu muss zuerst getestet werden, ob der Baum die AVL-Eigenschaft hat. Es ist jedoch jederzeit möglich, das Kästchen zu deaktivieren und einen unbalancierten Baum zu erzeugen.

2.5.4. Baum auf AVL-Eigenschaft testen




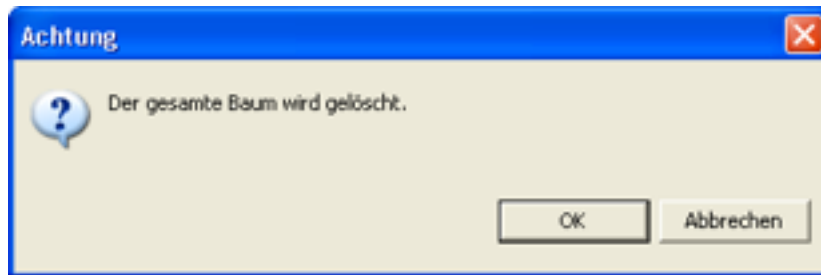
Hinweisfenster des AVL-Tests

Wenn der AVL-Modus einmal deaktiviert sein sollte, so ermöglicht das Programm einen Test des Baumes auf die AVL-Eigenschaft. Dabei erfolgt eine Berechnung und Anzeige der Balancen aller Knoten und das eventuelle Markieren von Knoten, deren Balance sich nicht mehr im Rahmen der AVL-Eigenschaft bewegt.

Es wird ein Hinweis-Dialog geöffnet, der Ihnen das Ergebnis des Tests präsentiert. Sollte der Baum tatsächlich die AVL-Eigenschaft besitzen, so wird Ihnen angeboten, direkt in den AVL-Modus zu wechseln.

2.5.5. Baum löschen

Mit einem Klick auf den Button  in der Werkzeugleiste können Sie nach einer Sicherheitsabfrage die gesamte Baumstruktur löschen und mit einer leeren Arbeitsfläche neu beginnen.



Sicherheitsabfrage beim Löschen des Baumes

2.6. Algorithmussteuerung

Aufgabe des Moduls **AVL-Bäume** ist es, Baumalgorithmen, wie das Einfügen und Löschen von Knoten, zu visualisieren. Jeder Algorithmus ist in verschiedene Teilschritte unterteilt, die nacheinander angezeigt werden. Das Visualisieren erfolgt dabei durch das Zeichnen des Baumes, durch die Erklärung der Schritte im Dokumentationsbereich und im Logbuch und durch die Neuberechnung der baumspezifischen Daten, die im Infobereich präsentiert werden. Nachdem Sie einen Algorithmus gestartet haben, verweilt er in einem Initialzustand und wartet auf Ihre Eingabe. Nun haben Sie die Möglichkeit, den Algorithmus in kleinen oder großen Schritten zu durchlaufen; Sie können ihn sofort beenden oder direkt abbrechen. Dafür bietet die Algorithmussteuerung die entsprechenden Werkzeuge.

2.6.1. Schritt-Pfeile

Mittels der Schritt-Pfeile steuern Sie die Abfolge der Einzelschritte und bekommen so eine detaillierte Sicht auf die Arbeitsweise des Algorithmus. Das Programm bietet Ihnen die Möglichkeit, einen Teilschritt rückgängig zu machen und damit gewisse Abläufe zu wiederholen. Die Schritt-Pfeile, welche die Rückgängigfunktion anbieten, weisen in ihrer Richtung nach links und sind dadurch intuitiv von den Vorwärts-Pfeilen zu unterscheiden.

Zusätzlich gibt es für jede Richtung einen großen und einen kleinen Schritt, der per Knopfdruck ausgeführt wird.

Kleine Schritte beim Einfügen eines Knotens stellen Schlüsselvergleiche, Balancenberechnungen und Rotationen dar. Große Schritte hingegen sind zum Beispiel das Suchen der Einfügestelle, das Einfügen an dieser und die gesamte Balancenaktualisierung.

Einzel-Schritt-Pfeile

Ein Klick auf diese Buttons realisiert einen kleinen Algorithmusschritt zurück bzw. nach vorn.

2. Das Modul AVL-Bäume

⏮ ⏭ Block-Schritt-Pfeile

Ein Klick auf diese Buttons realisiert einen großen Schritt zurück bzw. nach vorn. Sollte der Algorithmusablauf an eine Stelle geraten, an der es nur noch einen kleinen Schritt nach vorn bzw. zurück gibt, so hat der Block-Schritt die selbe Funktionalität wie ein Einzel-Schritt.

2.6.2. 🚫 🏁 Abbruch und Beenden-Buttons

Klicken Sie auf den Beenden-Button 🏁 um den laufenden Algorithmus bis zum Ende auszuführen.

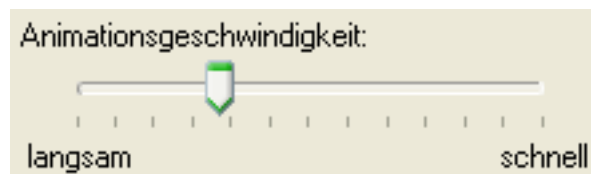
Klicken Sie auf den Abbruch-Button 🚫 um den laufenden Algorithmus abzubrechen. Der Baum hat danach den gleichen Status wie vor Beginn des Algorithmus.

Ist ein Algorithmus beendet, so steht Ihnen diese Option nicht mehr zur Verfügung, weil nur der *laufende* Algorithmus abgebrochen werden kann.

2.6.3. Animationsgeschwindigkeit

Beim Generieren eines Zufallsbaumes haben Sie die Option, den Ablauf der Baumerzeugung als Animation ablaufen zu lassen. Starten Sie in diesem Modus, so beginnt die Animation sofort und kann mit dem Geschwindigkeitsregler schneller oder langsamer abgespielt werden. Zu Beginn steht dieser auf der mittleren Position. Verschieben Sie den Regler nach links, um die Animation zu verlangsamen bzw. nach rechts, um sie zu beschleunigen.

Eine Animation der anderen Algorithmusabläufe ist in dieser Version von **AVL-Bäume** nicht integriert.



Der Regler für die Animationsgeschwindigkeit

2.7. Dokumentation

Da das Modul **AVL-Bäume** vor allem zu Lehr- und Lernzwecken eingesetzt werden soll, ist eine detaillierte Dokumentation der Algorithmen unumgänglich. Für die Einzelheiten des Algorithmustextes steht ein Auszug aus dem Vorlesungsskript von Prof. Vogler zur Verfügung. Ein Logbuch in der rechten unteren Ecke des Bildschirms führt Protokoll über den Stand und die Beschaffenheit des einzelnen Algorithmusteilschrittes.

Zu guter Letzt wird ein Infobereich angeboten, in dem wichtige Baumdaten zusammengefasst sind.

2.7.1. Skript

Der Dokumentationsbereich, der das Skript enthält, befindet sich am unteren Bildschirmrand. Es handelt sich hierbei um einen Auszug des Skripts zur Vorlesung „Algorithmen und Datenstrukturen“ von Prof. Vogler (TU Dresden), Version vom 2. Oktober 2003. Im Rahmen dieser

Vorlesung soll das Modul vorwiegend eingesetzt werden.

Bei dem jeweils aktuellen Algorithmustext handelt es sich um die Aktion, die als nächstes im Ablauf des Algorithmus erfolgen wird. Sie wird rot markiert angezeigt.

2.7.2. Logbuch

Das Logbuch ist eine weitere Möglichkeit, den Ablauf des Algorithmus zu verfolgen. Es bezieht sich in erster Linie auf baumspezifische Daten und verwendet zum Beispiel konkrete Schlüsselwerte, anhand deren die Aktionen des Algorithmus besser verstanden werden sollen.

Auch hier wird der aktuelle Eintrag rot markiert dargestellt. Dieser bezieht sich aber auf die zuletzt ausgeführte Aktion.

2.7.3. Infobereich

Der Infobereich ist hauptsächlich dafür gedacht, Ihnen schnell wichtige Daten des Baumes bereit zu stellen. Hier finden Sie folgende Punkte:

- **ANZAHL DER KNOTEN**
Dieser Punkt fasst für Sie die Anzahl der Knoten im Baum zusammen.
- **BAUMHÖHE**
Hier finden Sie die Anzahl der Level des Baumes.
- **DURCHSCHNITTLLICHE SUCHTIEFE**
Dieser Wert berechnet sich durch die Summe der Level aller Knoten geteilt durch die Anzahl dieser. Der Wert ist ein Indiz dafür, wie gut der Baum ausbalanciert ist bzw. wie groß der Suchaufwand im Durchschnitt ist.

2.8. Zusatzfunktionen

Dieses Kapitel widmet sich den Easteregg des Moduls **AVL-Bäume**.

Sollten Sie sich lieber selber gerne auf die Suche nach diesen Zusatzfunktionen machen wollen, so überspringen Sie besser dieses Kapitel.

Für alle Anderen folgt nun eine Übersicht zum Baumnavigator und dem Beamermodus.

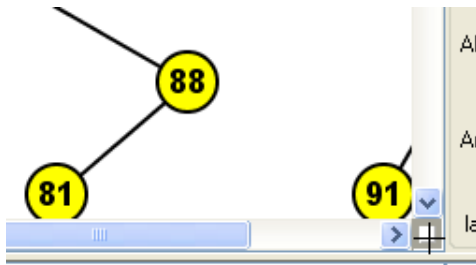
2.8.1. Navigator

Der Navigator ist eine kleine, versteckte Zusatzfunktion, die die Arbeit mit großen Bäumen erheblich vereinfachen kann. Er stellt eine willkommene Hilfe für das Scrollen der Zeichenfläche dar, ist aber nicht so einfach zu finden.

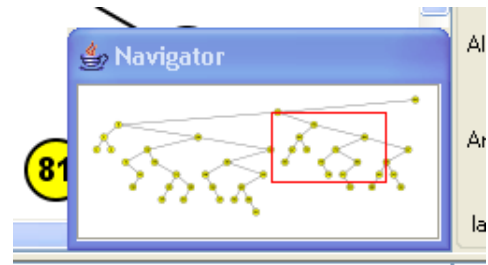
- Wenn der Baum, der auf der Zeichenfläche angezeigt wird, zu groß für diese wird, so erscheinen Schiebebalken, mit denen Sie den Bildausschnitt verschieben können.
- Klicken Sie nun auf das kleine Quadrat in der rechten unteren Ecke der Zeichenfläche, genau zwischen den beiden Schiebebalken. Halten Sie dabei die linke Maustaste gedrückt.

2. Das Modul AVL-Bäume

- Eine kleine Übersichtskarte des Baumes mit einem Ausschnittfenster erscheint. Bewegen Sie die Maus (mit gedrückter Taste) und das Ausschnittfenster, das den Bildschirminhalt der Zeichenfläche repräsentiert, folgt Ihren Bewegungen.



Ein Klick auf das kleine Kästchen...

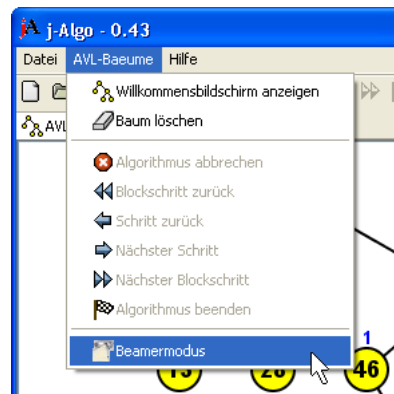


...öffnet den Navigator!

2.8.2. Beamermodus

Der Beamermodus ist in erster Linie für die Präsentation in Vorlesungen oder ähnlichen Veranstaltungen gedacht. Ist dieser Modus aktiv, so werden die Knoten des Baumes und die Einträge des Logbuches vergrößert dargestellt. Der Algorithmustext aus dem Skript von Prof. Vogler bleibt dabei unverändert, weil davon ausgegangen wird, dass die interessierten Studenten der Vorlesung über ein (eventuell aktuelleres) Skript verfügen.

Sie erreichen den Beamermodus über den Menüpunkt **<AVL-BÄUME>** → **<BEAMERMODUS>**. Ist der Modus aktiv, so erscheint neben diesem Menüeintrag ein Häkchen. Um den Modus wieder auszuschalten, entfernen Sie einfach den Haken per Klick.



Das Menü **<AVL-BÄUME>** mit dem Eintrag **<BEAMERMODUS>**

2.9. Impressum

Das Modul **AVL-Bäume** wurde im Sommersemester 2005 von der Praktikumsgruppe SWT05-PROG1 im Rahmen des externen Softwarepraktikums entwickelt. Mitwirkende waren die

Teammitglieder

- Alexander Claus — Chefprogrammierer
- Ulrike Fischer — Assistent
- Sebastian Pape — Administrator
- Jean Christoph Jung — Testverantwortlicher
- Matthias Schmidt — Sekretär

sowie der betreuende Tutor Marco Zimmerling.

Die Webseite des Projektes finden Sie unter <http://web.inf.tu-dresden.de/~swt05-p1/>.

Das Handbuch zu diesem Modul wurde erstellt von Matthias Schmidt und Jean Christoph Jung.

2. *Das Modul AVL-Bäume*

3. Das Modul Dijkstra

3.1. Einleitung

Das Modul **Dijkstra** visualisiert den bekannten Algorithmus von E. W. Dijkstra zum Finden der kürzesten Wege von einem Startknoten in einem Distanzgraphen. Der Algorithmus selbst ist unter anderem im Vorlesungsskript von Prof. Vogler „Algorithmen, Datenstrukturen und Programmierung“ zu finden. Aber auch im Internet existieren zahlreiche Quellen dazu.

Soweit es möglich gewesen ist, wurde beim Design des Moduls darauf geachtet, es weitgehend intuitiv und selbst-dokumentierend zu gestalten. Nichtsdestotrotz findet sich hier eine kurze Einführung in das **Dijkstra** - Modul.

3.2. Funktionsübersicht

Das Modul **Dijkstra** realisiert folgende Funktionen:

- graphisches Erstellen / Bearbeiten eines Distanzgraphen
- Erstellen / Bearbeiten eines Graphen mittels Kanten- / Knotenliste oder Adjazenzmatrix
- Speichern und Laden von Graphen
- Visualisierung des Dijkstra-Algorithmus

3.3. Modul starten

Um das Modul zu starten, wählt man im Menü <DATEI> das Submenü <NEU> und dann den Menübefehl **DIJKSTRA** . Im Hauptfenster erscheint nun die Oberfläche des **Dijkstra** - Moduls im Eingabe-Modus.

3.4. Symbolleiste

Die Symbolleiste stellt die Funktionen **SPEICHERN**, **SPEICHERN UNTER**, **RÜCKGÄNGIG** und **WIEDERHERSTELLEN** bereit.

3.5. Graph erstellen

Nach dem Start des Moduls wird die Oberfläche für das Erstellen eines Graphen angezeigt. Sie ist in die Bereiche

1. „Werkzeuge“
2. „Graph“
3. „Knotenliste“
4. „Kantenliste“
5. „Distanzmatrix“

aufgeteilt.

3.5.1. Graphische Eingabe/Erstellen eines Graphen per Maus

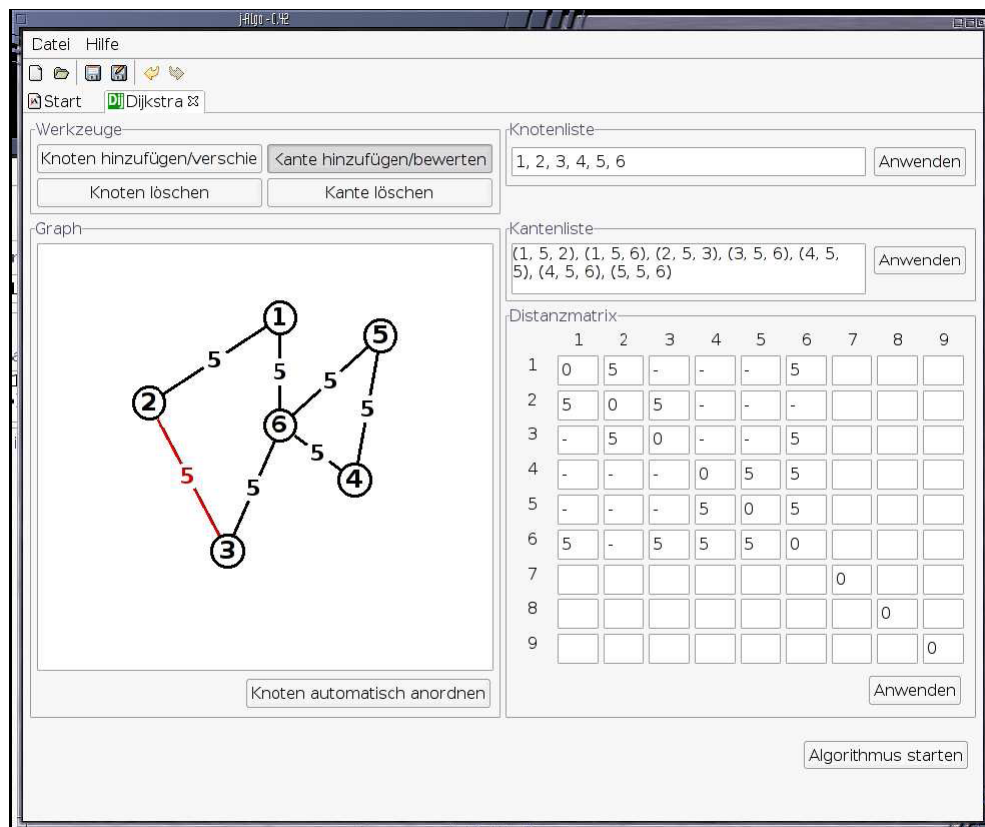
Das Erstellen eines Graphen per Maus wird durch die Werkzeuge

1. „Knoten hinzufügen/verschieben“ — Durch Klicken auf die Zeichenfläche wird ein neuer Knoten erzeugt. Ein bestehender Knoten kann durch Ziehen mit der Maus bewegt werden.
2. „Kante hinzufügen/bewerten“ — Indem man die Maus von einem Knoten zu einem anderen zieht, entsteht zwischen ihnen eine neue Kante. Die Kantenbewertung wird geändert, wenn man sie herauf- bzw. hinunterzieht.
3. „Knoten löschen“ — Ein angeklickter Knoten wird gelöscht.
4. „Kante löschen“ — Eine angeklickte Kante wird gelöscht.

unterstützt. Dabei geht man wie folgt vor: Nach der Auswahl des Werkzeugs „Knoten hinzufügen / verschieben“ kann man durch einfaches Klicken auf die weiße Zeichenfläche Knoten erstellen. Vorhandene Knoten können mit Drag&Drop verschoben werden.

Nachdem man alle Knoten angelegt hat, kann man nach Auswahl des Werkzeugs „Kante hinzufügen/bewerten“ den Graphen vervollständigen. Um eine Kante zu erstellen, klickt man erst den „Startknoten“ und dann den „Endknoten“ der Kante an. Es erscheint eine Kante zwischen den Knoten mit der Bewertung fünf. Diese Bewertung (auch Kantengewicht) kann verändert werden, indem man das Kantengewicht mit der Maus „festhält“ und nach oben (das Gewicht wird größer) oder unten (das Gewicht wird kleiner) zieht.

3.5. Graph erstellen



Graphisches Erstellen eines Graphen

3.5.2. Die Knotenliste

Die Knotenliste zeigt die Indizes aller Knoten durch Kommata getrennt. Durch Hinzufügen von Indizes werden auch neue Knoten erzeugt. Änderungen in der Knotenliste werden nach Betätigen der Schaltfläche „Anwenden“ übernommen.

3.5.3. Die Kantenliste

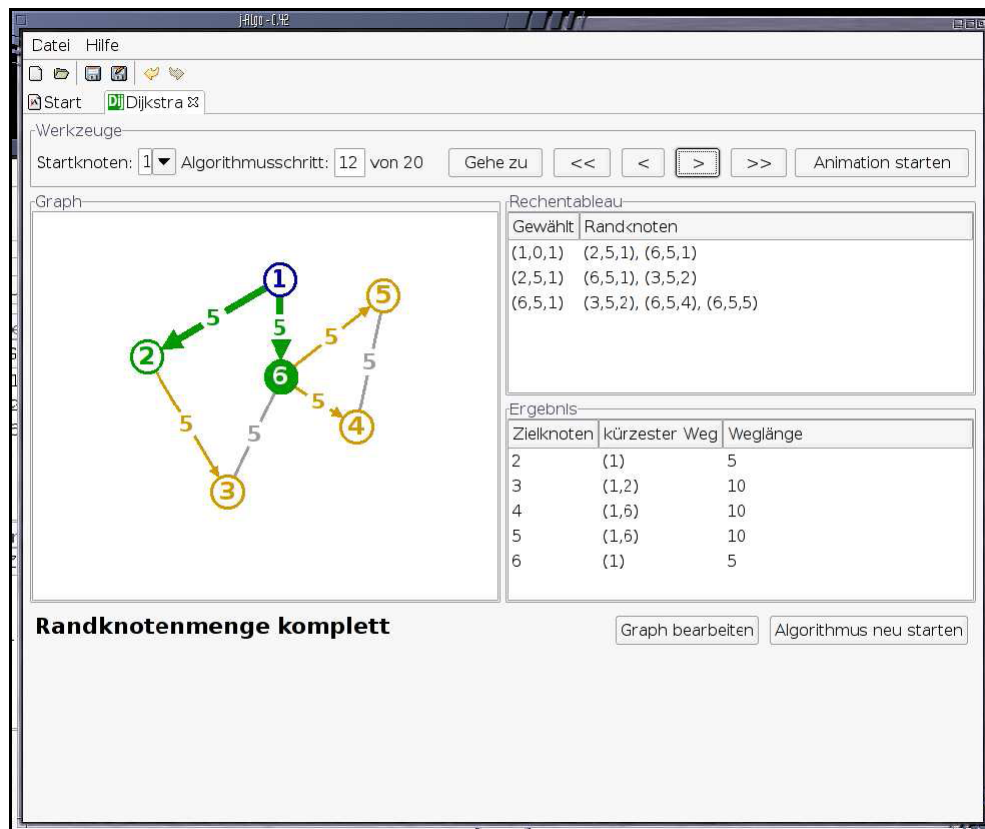
Die Kantenliste zeigt alle Kanten des Graphen im Format (VON, WEG, ZU). Durch Editieren dieser Liste können bestehende Kanten geändert und neue hinzugefügt werden. Auch hier ist zu beachten, dass Änderungen erst durch Klicken von „Anwenden“ übernommen werden.

3.5.4. Die Adjazenzmatrix

Die Adjazenzrelation des Graphen ist in dieser Matrix dargestellt. Kanten und Knoten können in jener durch einfaches Eingeben einer Kantenbewertung erzeugt werden. Dabei muss nicht beachtet werden, daß die Matrix symmetrisch bleibt, da dies automatisch gewährleistet wird. Nach dem Bearbeiten der Matrix darf nicht vergessen werden, den „Anwenden“ - Button zu betätigen, damit die Änderungen übernommen werden.

3. Das Modul Dijkstra

3.6. Ablauf des Algorithmus



Der Algorithmus läuft

Ist man mit dem Aussehen seines Graphen zufrieden, wird durch Auswählen des „Algorithmus starten“-Buttons in den Algorithmus-Modus des Moduls gewechselt. Der Graph erscheint nun zuerst grau und ändert sich farblich im weiteren Verlauf, um die verschiedenen Zustände des **Dijkstra** -Algorithmus darzustellen.

Um die einzelnen Schritte des Algorithmus abzuarbeiten, kann man man mit den „<“- und „>“-Buttons zum vorherigen bzw. nächsten Schritt springen. „«“ und „»“ überspringen gleich mehrere Schritte.

Falls man sich entschließt, einen anderen Graphen untersuchen zu wollen, kann mit „Graph bearbeiten“ in den Editiermodus zurückgekehrt werden. „Algorithmus neu starten“ springt zurück an den Anfang des Algorithmus.

3.7. Impressum

Das Modul **Dijkstra** wurde im Sommersemester 2005 von der Praktikumsgruppe SWT05-PROG2 im Rahmen des externen Softwarepraktikums entwickelt. Mitwirkende waren die

Teammitglieder

- Frank Staudinger — Chefprogrammierer
- Julian Stecklina — Assistent
- Hannes Straß — Administrator
- Martin Winter — Testverantwortlicher
- Steven Voigt — Sekretär

sowie der betreuende Tutor Marco Zimmerling.

Die Webseite des Projektes finden Sie unter <http://web.inf.tu-dresden.de/~swt05-p2/>.

3. *Das Modul Dijkstra*

A. Einleitung zu Datenstrukturen

Anmerkung: Der Autor der folgenden Seiten (Kapitel A, B und C dieses Anhangs) ist Jean Christoph Jung (Teammitglied AVL-Modul)

Eine häufige Anwendung auf großen Datenmengen ist das Suchen: Das Wiederfinden eines bestimmten Elements oder bestimmter Informationen aus einer großen Menge früher abgelegter Daten. Um die Suche zu vereinfachen, werden den (möglicherweise sehr komplexen) großen Datensätzen eindeutige Suchschlüssel (Keys) zugeordnet. Der Vergleich zweier solcher Schlüssel ist in der Regel viel schneller als der Vergleich zweier Datensätze. Wegen der Schlüsselzuordnung reicht es aus, alle vorkommenden Algorithmen nur auf den Schlüsseln zu betrachten; in Wirklichkeit verweisen erst die Schlüssel auf die Datensätze.

Eine grundlegende Idee ist nun, die Daten in Form einer Liste oder eines Feldes abzulegen. Diese Datenstruktur ist höchst einfach. Jedoch ist der Aufwand für das Suchen relativ hoch, nämlich $O(n)$. Genauer gesagt benötigt man für eine erfolglose Suche n Vergleiche (bei n Elementen in der Datenstruktur), denn man muss jedes Element überprüfen, und für eine erfolgreiche Suche durchschnittlich $(n + 1)/2$ Vergleiche durchführen, da nach jedem Element mit der gleichen Wahrscheinlichkeit gesucht wird.

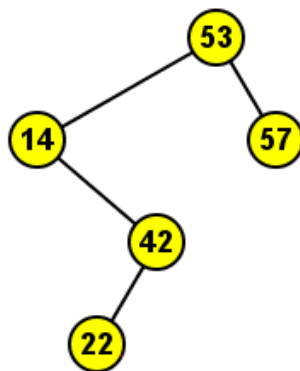
Eine Verbesserung dieses Verfahrens wäre, die Daten sortiert abzulegen, was zu einer binären Suche führt. Die Suche hat jetzt nur noch Komplexität $O(\log_2 n)$, da bei jedem Suchschritt das Feld halbiert wird. Der Nachteil ist, dass durch die Sortierung das Einfügen erschwert wird, da unter Umständen viele Datensätze bewegt werden müssen. Das Verfahren sollte also nur angewandt werden, wenn sehr wenige oder gar keine Einfügeoperationen ausgeführt werden müssen. Dann können die Daten anfangs mit einem schnellen Verfahren sortiert werden und müssen danach nicht mehr verändert werden.

Eine weitere Datenstruktur, die der Suchbäume, wollen wir hier vorstellen.

A. Einleitung zu Datenstrukturen

B. Suchbäume

Suchbäume sind binäre Bäume (jeder Knoten hat höchstens 2 Kinder) mit der Eigenschaft: Für jeden Knoten gilt: alle Schlüssel im rechten Teilbaum sind größer als der eigene Schlüssel und alle Schlüssel im linken Teilbaum sind kleiner. Diese Eigenschaft wird hier immer Suchbaumeigenschaft genannt.



Ein Beispiel für einen Suchbaum

An dieser Stelle noch eine Bemerkung zu den Schlüsseln: Die Schlüssel können Elemente einer beliebigen Menge sein, unter der Bedingung, dass auf dieser Menge eine Ordnungsrelation definiert ist. Die Ordnungsrelation wird offensichtlich für die Suchbaumeigenschaft benötigt, da dort die Begriffe „kleiner“ und „größer“ vorkommen. Eine häufig verwendete Menge sind die natürlichen Zahlen mit ihrer normalen Ordnungsrelation \leq . Alle Operationen auf Suchbäumen kann man sich also anhand der natürlichen Zahlen vorstellen.

Aus der Suchbaumeigenschaft kann man sich leicht rekursive Algorithmen für das Einfügen und Suchen in einem Suchbaum herleiten:

Algorithmus 1 *(Suchen eines Schlüssels s in einem Suchbaum)*

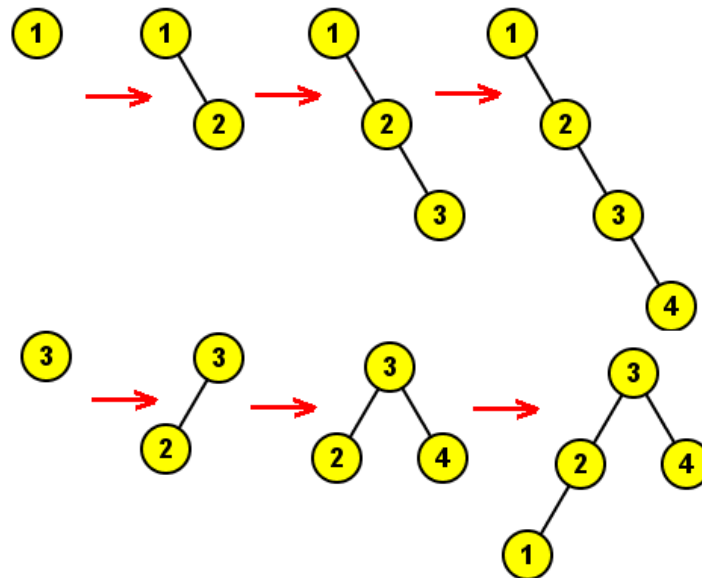
1. Falls Teilbaum leer, dann Schlüssel nicht im Baum vorhanden
2. Falls s gleich dem Schlüssel des aktuellen Knoten, dann Suche erfolgreich.
3. Falls s größer als Schlüssel des aktuellen Knotens, dann suche (rekursiv) s im rechten Teilbaum.
4. Falls s kleiner als Schlüssel des aktuellen Knotens, dann suche (rekursiv) s im linken Teilbaum.

B. Suchbäume

Algorithmus 2 (Einfügen eines Schlüssels s in einen Suchbaum)

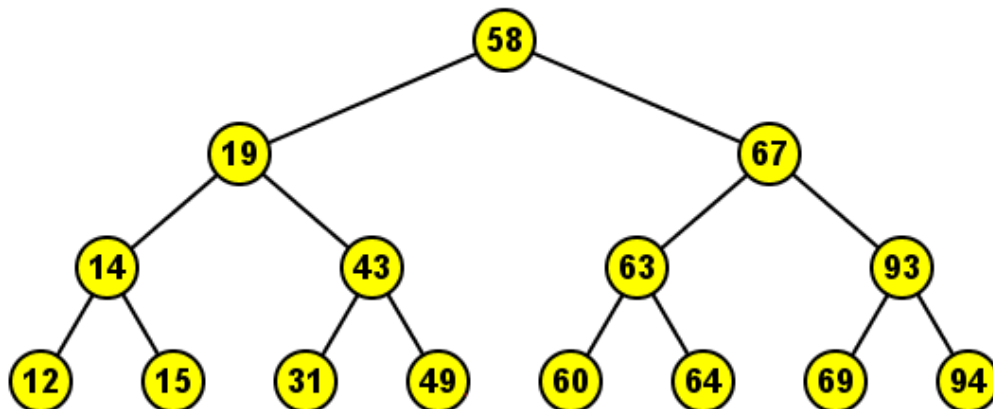
1. Falls Teilbaum leer, dann neuen Schlüssel hier einfügen.
2. Falls s gleich dem Schlüssel des aktuellen Knotens, dann Schlüssel bereits vorhanden, Einfügen nicht nötig.
3. Falls s größer als Schlüssel des aktuellen Knotens, dann füge (rekursiv) s in den rechten Teilbaum ein.
4. Falls s kleiner als Schlüssel des aktuellen Knotens, dann füge (rekursiv) s in den linken Teilbaum ein.

Mit Algorithmus 2 ergibt sich folgende Eigenschaft der Struktur von Suchbäumen: im Gegensatz zur sortierten Liste hängt die Struktur eines Baumes davon ab, in welcher Reihenfolge die Elemente eingefügt werden. So erhält man verschiedene Bäume, wenn man 1, 2, 3, 4 in dieser Reihenfolge und in der Reihenfolge 3, 2, 4, 1 einfügt.



Unterschiedliche Suchbäume bei unterschiedlicher Einfügereihenfolge

Im ersten Fall erhält man einen Suchbaum, der zur linearen Liste entartet ist. Das ist nicht nur in dieser speziellen Reihenfolge so, es gibt viele Möglichkeiten einen entarteten Baum zu erzeugen. Der Algorithmus hat also zwei Nachteile: zum einen kann der Suchaufwand linear zur Anzahl der Knoten im Baum sein, was keine Verbesserung zur linearen Liste darstellt; zum anderen hängt die Güte des Verfahrens von der Eingabefolge ab. Der Vorteil von Suchbäumen wird klar, wenn man einen „vollen“ Baum betrachtet, d.h. alle Pfade zu Blättern haben dieselbe Länge. Dann hat sowohl das Suchen (unabhängig davon, ob der Schlüssel im Baum vorhanden ist) als auch das Einfügen eine Komplexität von $O(\log n)$.

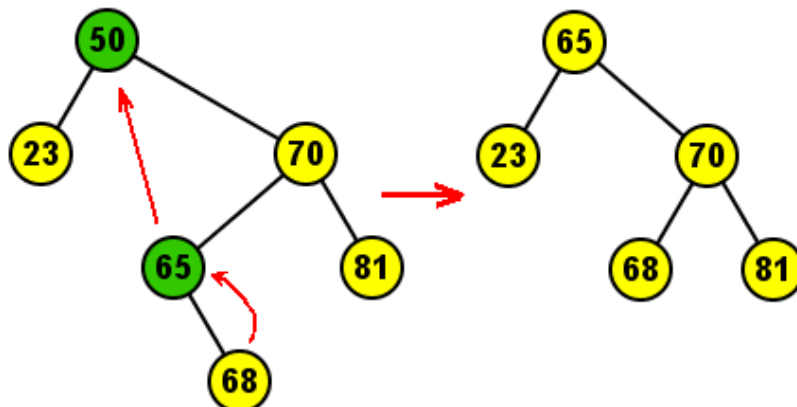


Ein Beispiel für einen vollen Baum

Es gibt einige Algorithmen, bei denen der Einfügealgorithmus so modifiziert ist, dass die entarteten Fälle vermieden werden und immer nahezu volle Bäume entstehen. Einen davon, den Algorithmus nach Adelson-Velskij und Landis (AVL), werden wir später betrachten.

Doch zunächst wollen wir noch eine weitere wichtige Operation auf Suchbäumen untersuchen: das Löschen. Leider ist es nicht ganz so einfach wie Suche und Einfügen.

Zuerst muss der zu löschende Schlüssel gesucht werden. Ist der betreffende Knoten ein Blatt, kann er einfach entfernt werden. Hat der zu löschende Knoten nur ein Kind, kann er durch dieses ersetzt werden. Der schwierige Fall ist, wenn er zwei Kinder hat. Damit die Suchbaumeigenschaft erhalten bleibt, muss man ihn durch den nächst größeren Schlüssel ersetzen. Der nächst größere Schlüssel befindet sich offensichtlich im rechten Teilbaum.



Löschen eines Knoten im Suchbaum

Nach dem Ersetzen bleibt die Suchbaumeigenschaft erhalten, weil alle Schlüssel aus dem linken Teilbaum ohnehin kleiner sind als die aus dem rechten. Außerdem sind auch alle Schlüssel aus dem rechten Teilbaum größer, sonst wäre es nicht der nächst größere Schlüssel gewesen. Aus diesen Überlegungen erhält man folgende verbale Beschreibung des Löschen-Algorithmus:

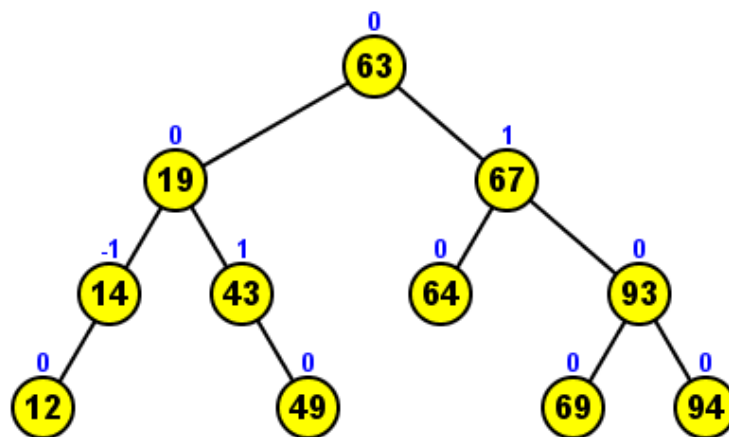
Algorithmus 3 (*Löschen eines Schlüssels s aus einem Suchbaum*)

1. Suche s nach Algorithmus 1. Falls s nicht im Baum enthalten, terminiert der Algorithmus.
2.
 - a) Ist der zu löschende Knoten ein Blatt, dann entferne ihn einfach aus dem Baum.
 - b) Hat der zu löschende Knoten nur ein Kind, dann ersetze ihn durch dieses.
 - c) Sonst suche den kleinsten Schlüssel im rechten Teilbaum: Gehe zum rechten Kind und dann immer zum linken Teilbaum, solange dieser nicht leer ist. Ersetze den zu löschenden Schlüssel durch den des so gefundenen Knotens. Ersetze den gefundenen Knoten durch sein rechtes Kind.

Man kann anstelle des nächst größeren Schlüssels genausogut den nächstkleineren nehmen, der Algorithmus funktioniert trotzdem. Zur Komplexität ist zu sagen, dass der Algorithmus maximal h Vergleiche macht, wobei h die Höhe des Baumes ist. Auch hier ist also die Komplexität abhängig von der Struktur des Baumes.

C. AVL-Bäume

AVL-Bäume (benannt nach Adelson-Velskij und Landis) sind spezielle Suchbäume: In jedem Knoten unterscheiden sich die Höhen des linken Teilbaums und des rechten Teilbaums um höchstens 1. Um diese Eigenschaft (AVL-Eigenschaft) abzusichern, wird für jeden Knoten ein Balancefaktor eingeführt. Der Balancefaktor ist die Differenz der Höhen des rechten Teilbaums und des linken Teilbaums. Also gilt für jeden AVL-Baum, dass alle Balancefaktoren aus $\{-1, 0, 1\}$ sind. Bäume mit AVL-Eigenschaft sind niemals als lineare Liste entartet (sofern sie denn mehr als 2 Knoten haben), sondern sind immer fast vollständig. Zwischen der Höhe h eines Baumes und der Anzahl n seiner Knoten besteht folgender Zusammenhang: $h \leq 2 \cdot \log_2 n$. Das bedeutet, dass für das Suchen logarithmische Komplexität garantiert werden kann (das Suchen erfolgt gemäß Algorithmus 1).



Ein Beispiel für einen AVL-Baum mit Balancen

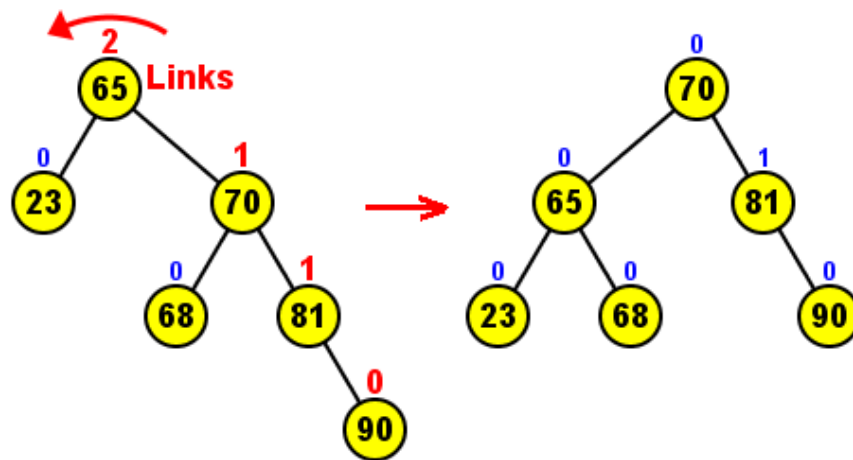
Jetzt muss noch untersucht werden, wie groß der zusätzliche Aufwand beim Einfügen ist, um die AVL-Eigenschaft zu wahren. In jedem Fall wird der neue Knoten als Blatt eingefügt (nach demselben Algorithmus wie bei Suchbäumen). Dabei kann sich der Balancefaktor ändern. Allerdings kann man sich leicht klarmachen, dass das nur entlang des Suchpfades passieren kann. Die Aktualisierung von Balancefaktoren erfolgt von der Einfügestelle zur Wurzel. Hier der Algorithmus:

C. AVL-Bäume

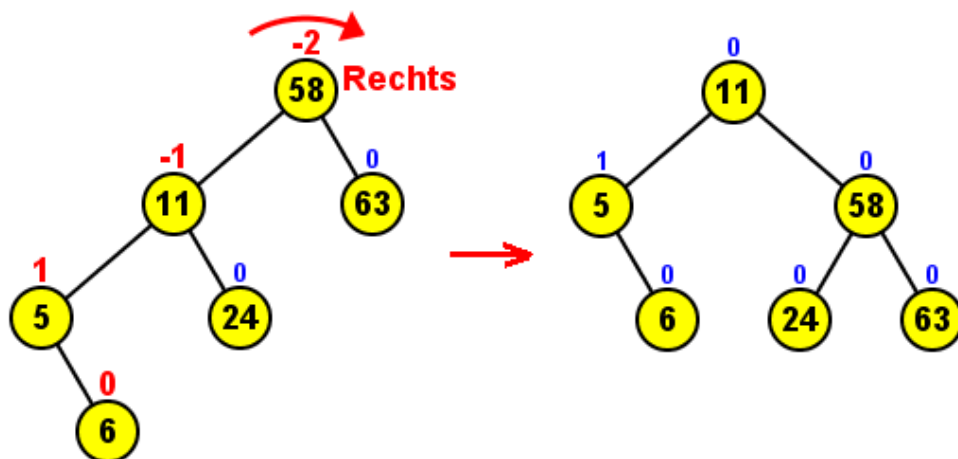
Algorithmus 4 (Algorithmus zum Einfügen eines Elements x in einen AVL-Baum)

1. Füge das neue Element x als direkten Nachfolger des Knotens n als Blatt ein, sodass die Suchbaumeigenschaft erfüllt bleibt. Aktualisiere $n.balance$.
2. Setze n auf den Vorgängerknoten von n .
 - a) Falls x im linken Unterbaum von n eingefügt wurde
 - i. wenn $n.balance == 1$ dann $n.balance = 0$ und gehe nach 3.
 - ii. wenn $n.balance == 0$, dann $n.balance = -1$ und gehe nach 2.
 - iii. wenn $n.balance == -1$ und
 - wenn $n.left.balance == -1$, dann Rechts(n)-Rotation.
 - wenn $n.left.balance == 1$ dann Links($n.left$)-Rechts(n)-Rotation.
 Gehe zu 3.
 - b) Falls x im rechten Unterbaum von n eingefügt wurde
 - i. wenn $n.balance == -1$ dann $n.balance = 0$ und gehe nach 3.
 - ii. wenn $n.balance == 0$, dann $n.balance = 1$ und gehe nach 2.
 - iii. wenn $n.balance == 1$ und
 - wenn $n.left.balance == 1$, dann Links(n)-Rotation.
 - wenn $n.left.balance == -1$ dann Rechts($n.left$)-Links(n)-Rotation.
 Gehe zu 3.
3. Gehe zurück zur Wurzel.

Zur Analyse dieses Algorithmus: Das reine Einfügen erfolgt gemäß Einfügen im Suchbaum (Algorithmus 2), allerdings ist hier sichergestellt, dass sich die Höhe logarithmisch zur Anzahl der Knoten verhält, d.h. auch der Aufwand für das Einfügen ist garantiert logarithmisch. Wie gesagt können sich jedoch Balancefaktoren geändert haben, sodass die AVL-Eigenschaft nicht mehr erfüllt ist. Das wird durch sogenannte Rotationen behoben. Es gibt zwei Typen von Rotationen – Linksrotation und Rechtsrotation, jeweils um einen Knoten n .



Linksrotation um den Knoten 65



Rechtsrotation um den Knoten 58

Falls durch das Einfügen irgendwo ein Balancefaktor 2 (-2) entsteht (größere Änderungen können beim Einfügen eines Knotens offensichtlich nicht auftreten), heißt das, dass sich im rechten (linken) Teilbaum die Höhe um 1 erhöht hat. Durch Rotation(en) wie im Algorithmus angegeben, wird aber genau diese Höhe wieder reduziert. Somit ist klar, dass man maximal zweimal rotieren muss, der Aufwand ist also noch erträglich, im Gegensatz zum Löschen, wie man gleich sehen wird.

Genauso wie Einfügen verändert auch Löschen eines Knotens aus einem AVL-Baum die Balancefaktoren, also müssen auch hier Rotationen ausgeführt werden. Hier der Algorithmus:

Algorithmus 5 *Löschen eines Knotens aus einem AVL-Baum)*

1. Lösche den Knoten analog zum Löschen im Suchbaum (Algorithmus 3). Falls der Knoten ein Blatt war oder nur einen linken Nachbarn hatte, setze aktuellen Knoten auf den Vater. Sonst setze aktuellen Knoten auf den Vater des Knotens mit dem nächst größeren Schlüssel.
2. Berechne den Balancefaktor des aktuellen Knotens neu. Falls
 - a) Balance 2 und rechte Balance -1, dann Rechts($n.right$)-Links(n)-Rotation.
 - b) Balance 2 und rechte Balance nicht -1, dann Links(n)-Rotation.
 - c) Balance -2 und linke Balance 1, dann Links($n.left$)-Rechts(n)-Rotation.
 - d) Balance -2 und linke Balance nicht 1, dann Rechts(n)-Rotation.
 - e) sonst keine Rotation.

Wiederhole diesen Schritt solange, bis die Wurzel erreicht ist.

Auch hier wollen wir den Aufwand des Algorithmus etwas genauer untersuchen. Schritt 1 entspricht dem Löschen aus dem Suchbaum, nur garantiert mit logarithmischem Aufwand. Die Frage ist nun, ob, wie beim Einfügen, der Algorithmus mit maximal zwei Rotationen auskommt. Leider ist das nicht der Fall. Das liegt an der erwähnten Eigenschaft der Rotationen, sie verringern die Höhe eines Teilbaums. Beim Einfügen war das gut, da durch das Anhängen

C. AVL-Bäume

eines Knotens gerade die Höhe vergrößert wurde. Hier jedoch ist das unvorteilhaft, es kann passieren, dass man mehrere Rotationen auf dem Weg zur Wurzel durchführen muss. Die Anzahl der Rotationen ist nur durch die Höhe des Baums beschränkt. Das ist in Anwendungsfällen nicht wünschenswert.

Bei zeitkritischen Anwendungen muss man also entweder auf einen anderen Algorithmus ausweichen, oder Varianten wie etwa Lazy-Delete implementieren, d.h. der Knoten wird nur als gelöscht markiert und wird später (wenn Zeit ist) aus dem Baum entfernt.

Literaturverzeichnis

- [1] R. Sedgewick, „Algorithmen in C++“, Addison-Wesley, 5. Auflage, 1999
- [2] Prof. Vogler, „Vorlesungsskript Algorithmen, Datenstrukturen und Programmierung“, TU Dresden, 2003
- [3] <http://de.wikipedia.org/wiki/AVL-Baum>
- [4] <http://dbs.uni-leipzig.de/de/skripte/ADS1/HTML/kap6-11.html>