Inhaltsverzeichnis

1 Implementierung und Laufzeittests

In diesem Kapitel wird kurz die Implementierung zum Tango Baum beschrieben und dann werden noch die Laufzeittests dargestellt.

1.1 Implementierung

Implementiert wurde ein Tango Baum, ein Rot Schwarz Baum in der Rolle als Hilfsstruktur für den Tango Baum. Außerdem wurde die *access* Operation des Splay Baum implementiert, um Laufzeittest zwischen diesem und dem Tango Baum durchführen zu können. Bedient werden kann das Programm, über eine einfach gehaltene graphische Oberfläche. Das Programm wurde mit Java 8 übersetzt und als IDE wurde Apache NetBeans 12.0 verwendet. Abbildung 5 stellt ein Klassendiagramm dar.
Abbildung 1 zeigt das Hauptfenster. Oben ist ein Referenzbaum zu einem Tango Baum mit 15 Knoten dargestellt, unten der Tango Baum. Preferred Childs und die Wurzeln von Hilfsbäumen sind grün dargestellt.
Mit dem Menüpunkt „access“ wird das Fenster aus Abbildung 2 geöffnet. Mit diesem werden *access* Operationen angestoßen. Außerdem können die Bäume damit zurückgesetzt werden.
Mit dem Menüpunkt „RuntimeTest“ wird das Fenster aus Abbildung 3 geöffnet. Mit diesem werden Laufzeittests zwischen dem Tango Baum und dem Splay Baum angestoßen. Auf die Parameter und den Aufbau der Zugriffsfolgen, wird im Abschnitt zu den Laufzeittests eingegangen.

1.1.1 Beschreibung der Klassen

SplayTree und SplayNode Der Splay Baum startet genau wie der Tango Baum perfekt balanciert, auch wenn sich dies bei längeren Zugriffsfolgen

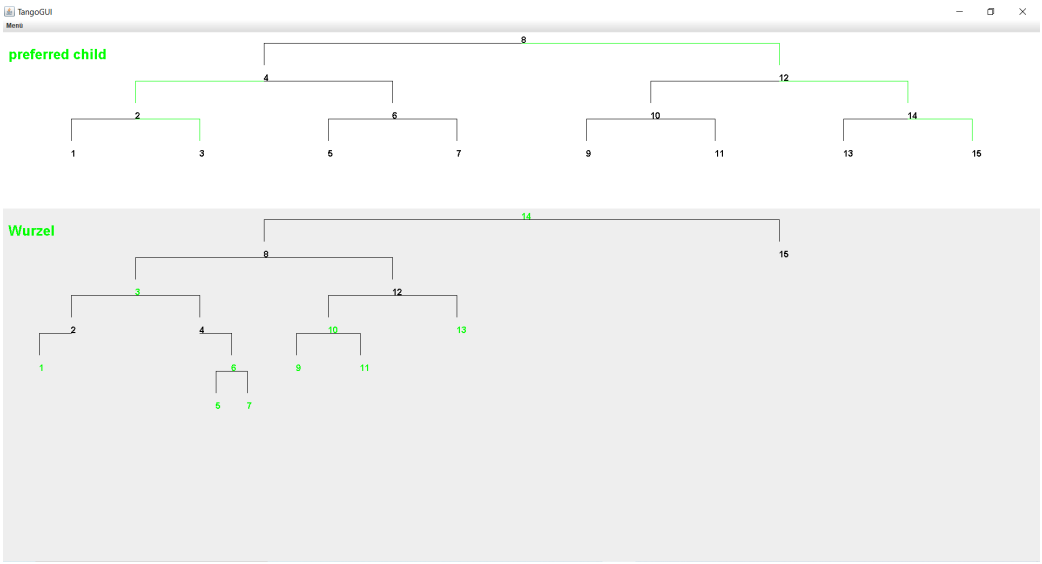

Abbildung 1: Oberfläche zum Tango Baum

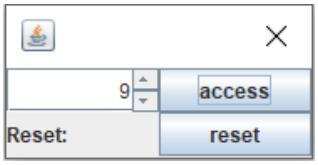

Abbildung 2: *access* Operationen anstoßen.

praktisch nicht auswirken sollte. Ansonsten gibt es keine Besonderheiten. *access* verhält sich genau wie im Kapitel zum Splay Baum beschrieben.

TangoNode Der TangoNode enthält bereits alle zwingend notwendigen Attribute eines Knoten im Tango Baum.

TangoAuxTree Klassen die als Hilfsstruktur im Tangobaum eingesetzt werden sollen, müssen diese Klasse erweitern. *setTree* wird benötigt, da die Klasse TangoTree die BST Struktur nur über das Attribut „auxTree“ erreicht. Gibt es eine Veränderung an der Wurzel des Tango Baum, wird die BST Struktur von „auxTree“ neu gesetzt. *updateDepthsPath* pflegt die Attribute „minDepth“ und „minDepth“ der TangoNode.

TangoTree „auxTree“ macht die Wurzel des Tango Baum erreichbar. Außerdem können über diesen Attribut die *split* und *join* Operationen aufgerufen werden. „auxTreeClass“ entspricht der Klasse der eingesetzten Hilfsbäu-

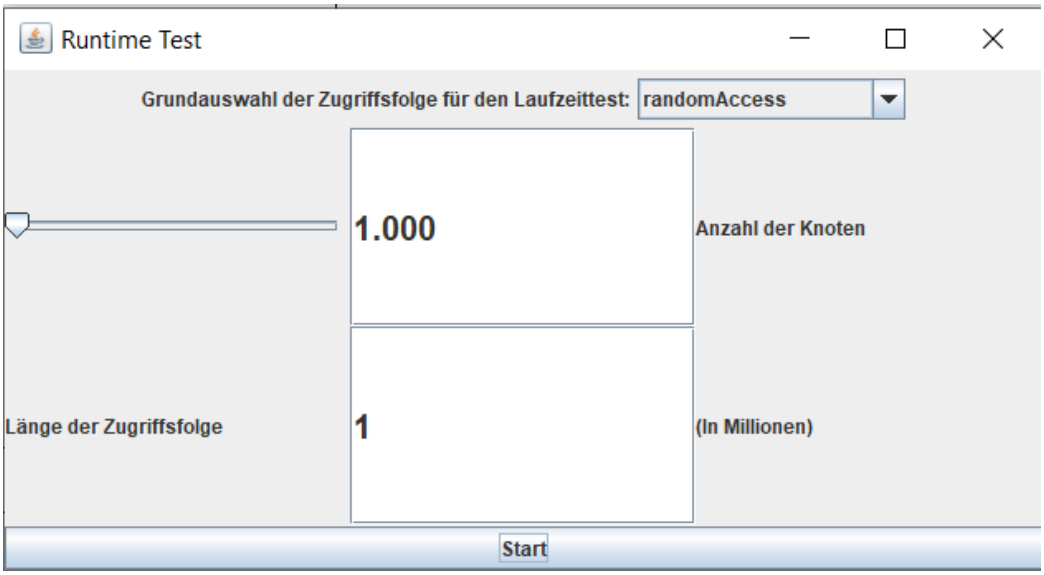

Abbildung 3: Laufzeittest anstoßen.

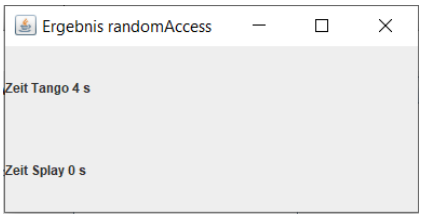

Abbildung 4: Ergebnisanzeige eines Laufzeittests.

men. Diese wird dem Constructor übergeben. Somit kann der RBT einfach durch eine andere geeignete Struktur ersetzt werden.

RedBlackTree und RBTNode Erweitern die abstrakten Klassen. RedBlackTree verhält sich wie im Kapitel zu RBT beschrieben.

RuntimeTest Hier ist die Durchführung der Laufzeittests umgesetzt. Mit „exit“ kann ein Test abgebrochen werden. Ein von dieser Klasse erzeugtes Objekt, führt genau einen Laufzeittest durch, die restlichen Attribute dienen dessen Parametrierung. Die Methoden führ die Test geben Arrays der Länge 2 zurück. Der erste Wert entspricht der Laufzeit des Tango Baum, der zweite der des Splay Baum. Um Programmabbrüchen aufgrund zu wenig Speicher vorzubeugen, wurde bei den Projekteigenschaften, die Option „ -Xmx4096m“ gesetzt, Abbildung 6. Ein Datenblatt zu dem verwendeten System ist dem Kapitel angefügt.

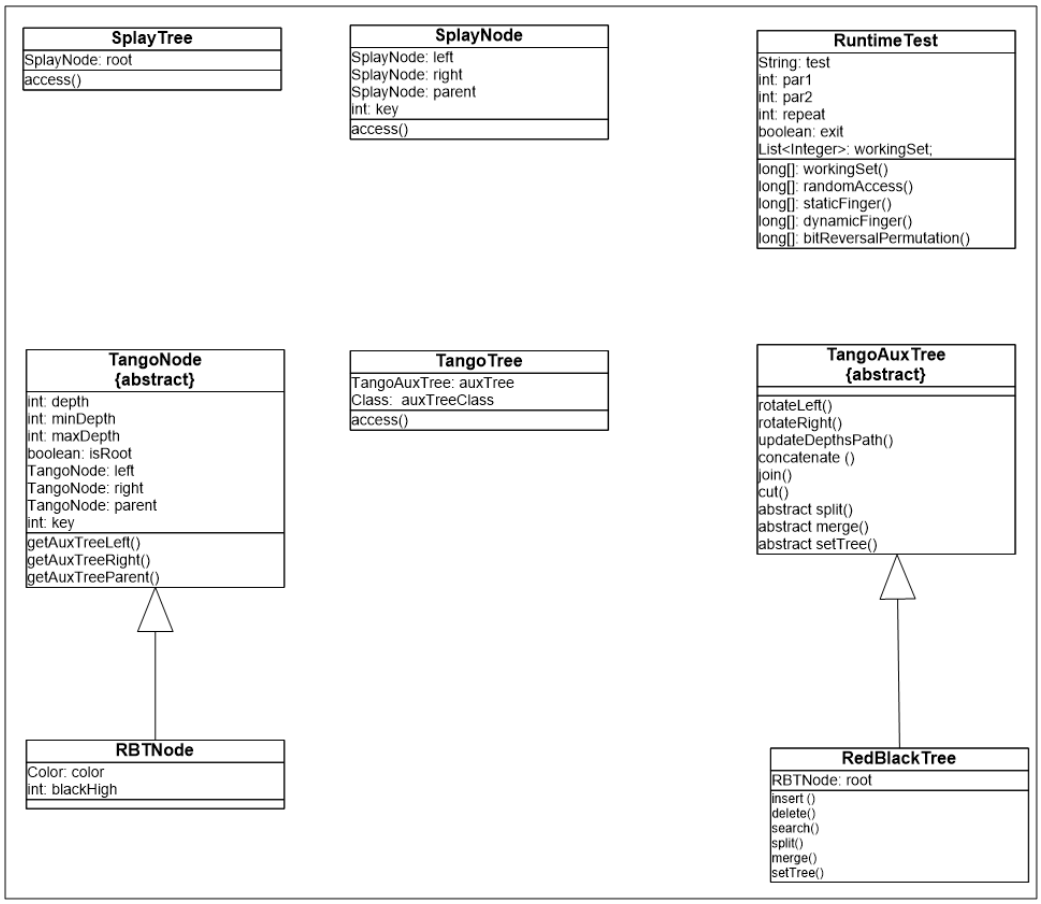

Abbildung 5: Wesentliche Klassen der Implementierung. Methoden zum direkten lesen bzw. schreiben von Attributen sind nicht dargestellt.

1.2 Laufzeittests zwischen Tango Baum und Splay Baum

Es werden Tests zu fünf Arten von Zugriffsfolgen durchgeführt. Zunächst wird immer der Aufbau der Zugriffsfolge beschrieben und dann die Ergebnisse präsentiert. n entspricht der Anzahl der Knoten, m der Länge der Zugriffsfolge. Die Schlüsselmengen haben immer die Form $\{1, 2, .., n\}$.

1.2.1 Zufällige Zugriffsfolge

Die Zugriffsfolge wird von einem Pseudozufallsgenerator erzeugt.

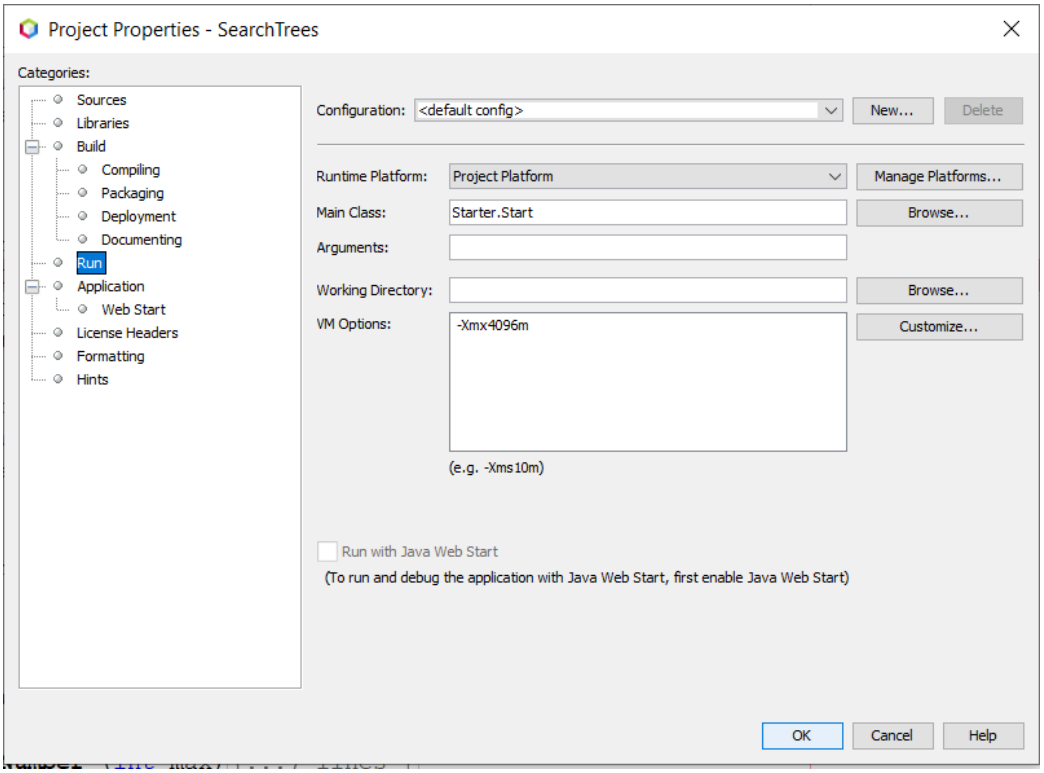

Abbildung 6: Zur Ausführung verwendbaren Speicher erweitert.

1.2.2 Bit reversal permutation

1.2.3 Static Finger

Sei $a = \lfloor n/2 \rfloor$. a ist der Parameter bei 2 Prozent der *access* Operationen. Auf $a+1$ und $a-1$ entfallen dann 1 Prozent (gemeinsam 2 Prozent) der restlichen *access* Operationen. Dieses vorgehen iteriert bis ein Prozent der Anzahl der verbleibenden *access* Operationen, weniger als 1 ergibt. Die Anordnung der Schlüssel in der Zugriffsfolge geschieht wieder über einen Pseudozufallsgenerator.

1.2.4 Dynamic Finger

Es wird die Zugriffsfolge $1, 3, 5, .., n-1, 1, 3, 5, .., n-1, ...$ verwendet.

1.2.5 Working Set

Das Working Set enthält 10 Prozent der in den Bäumen enthaltenen Schlüssel. Diese sind gleichmäßig über $1, 2, .., n$ verteilt. Eine beispielhaftes Working

Set wäre $\{1, 4, 7, .., n\}$, die verwendete Zugriffsfolge ist dann $1, 4, 7, .., n, 1, 4, 7, .., n, ...$

Literatur