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.

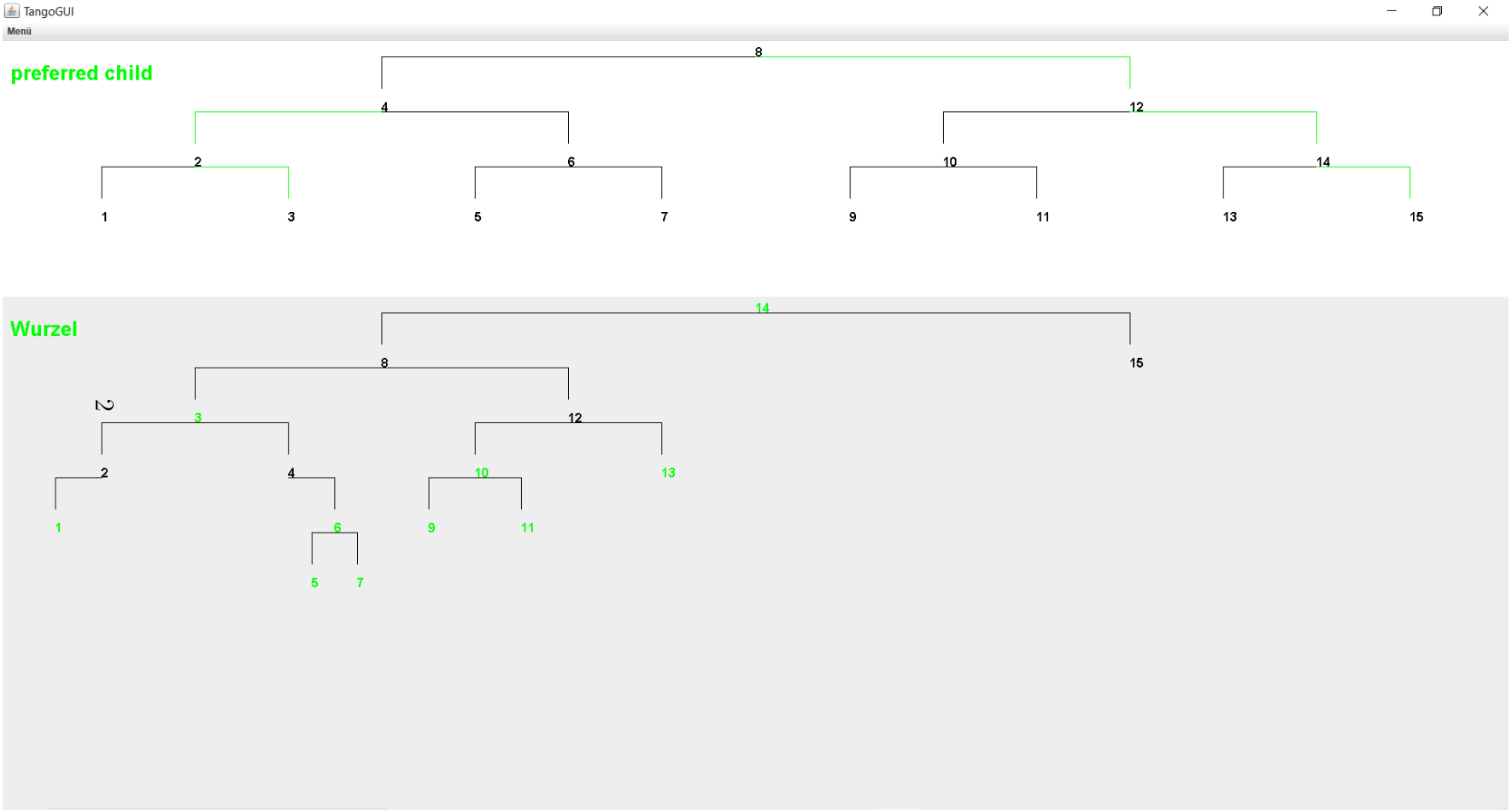

Abbildung 1: Oberfläche zum Tango Baum

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.

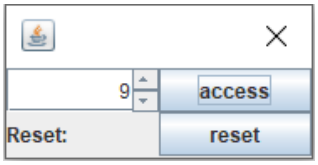

Abbildung 2: *access* Operationen anstoßen.

Mit dem Menüpunkt „access“ wird das Fenster aus Abbildung 2 geöffnet. Mit diesem werden *access* Operationen angestoßen. Außerdem können die Bäume damit zurückgesetzt werden.

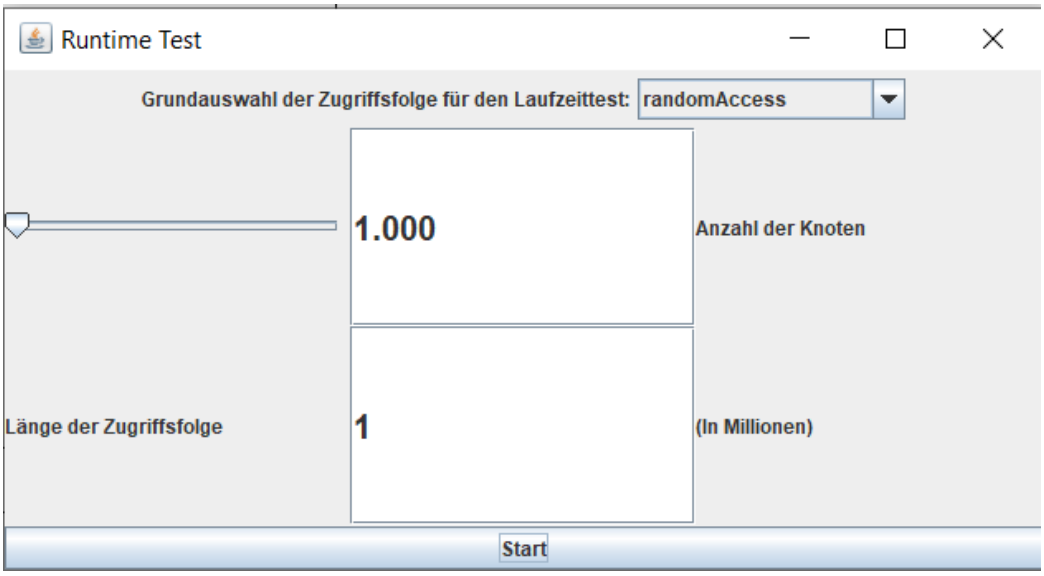

Abbildung 3: Laufzeittest anstoßen.

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.

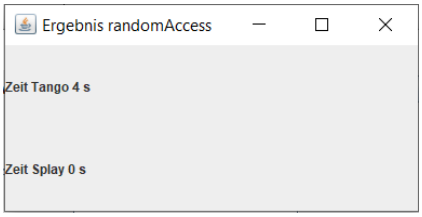

Abbildung 4: Ergebnisanzeige eines Laufzeittests.

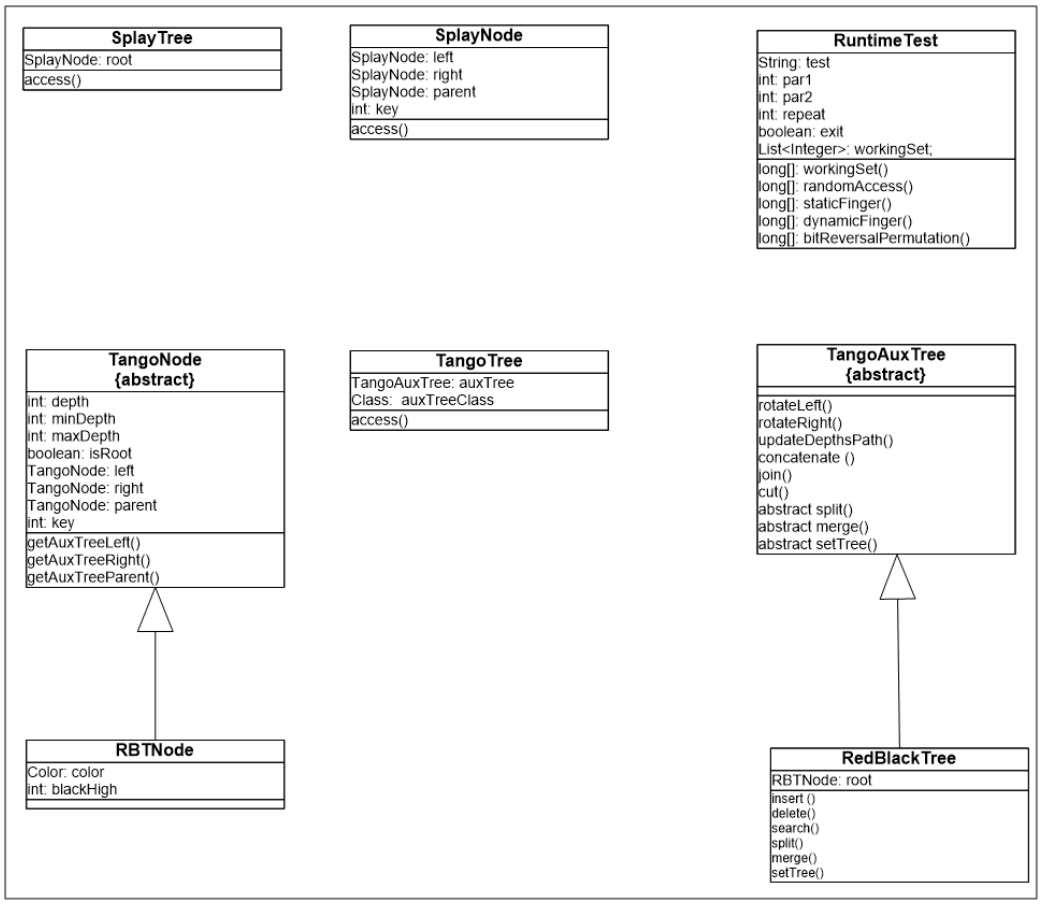

Abbildung 5: Wesentliche Klassen der Implementierung. Methoden zum direkten lesen bzw. schreiben von Attributen sind nicht dargestellt.

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

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 Hilfsbäumen. 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 für 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.

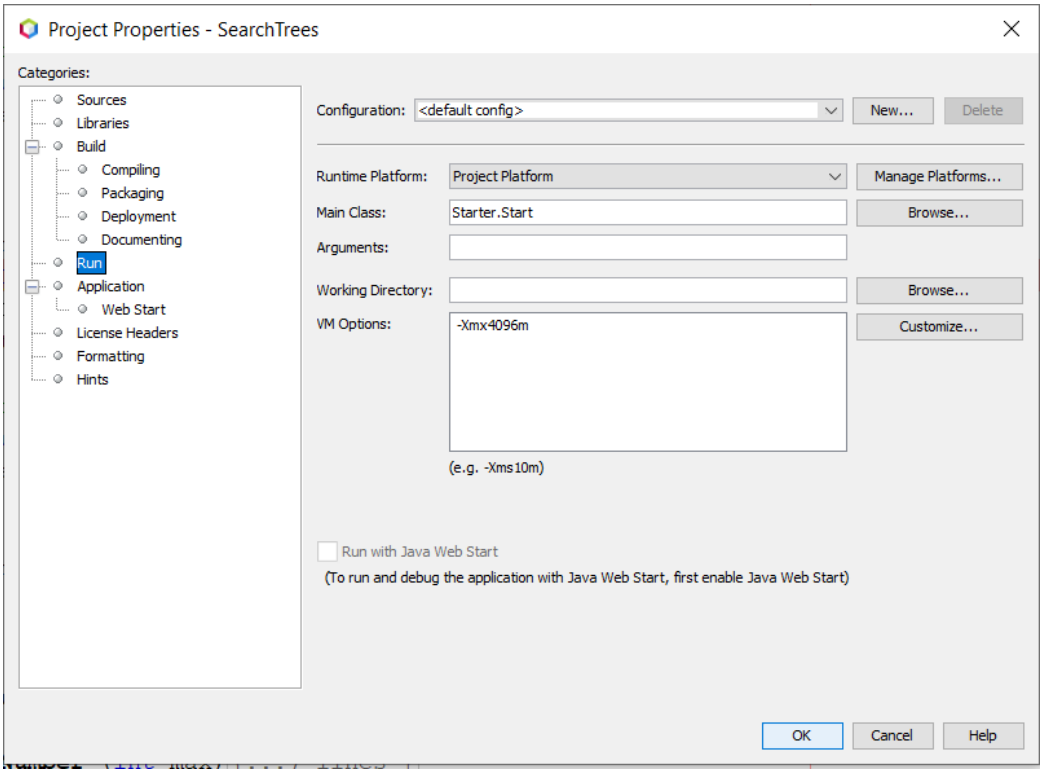

Abbildung 6: Zur Ausführung verwendbaren Speicher erweitert.

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 $\{1, 2, .., n\}$. Bei jeder Testart wurde zu jeder Knotenzahl, beim Tango Baum und Splay Baum immer exakt die gleiche Zugriffsfolge verwendet. m ist die Länge der Zugrissfolge. Gibt es keine Abhängigkeit von m zu n, so wurde für m immer 40 Millionen verwendet. Der Splay Baum wird durchweg die kürzeren Zeiten liefern. Eine mögliche Erklärung für die vergleichsweise oft hohen Zeiten des Tango Baum ist, dass dieser erst bei noch größeren Instanzen einen Vorteil, aus der Unterteilung in Hilfsbäume ziehen kann.

1.2.1 Zufällige Zugriffsfolge

Die Zugriffsfolgen werden von einem Pseudozufallsgenerator erzeugt.

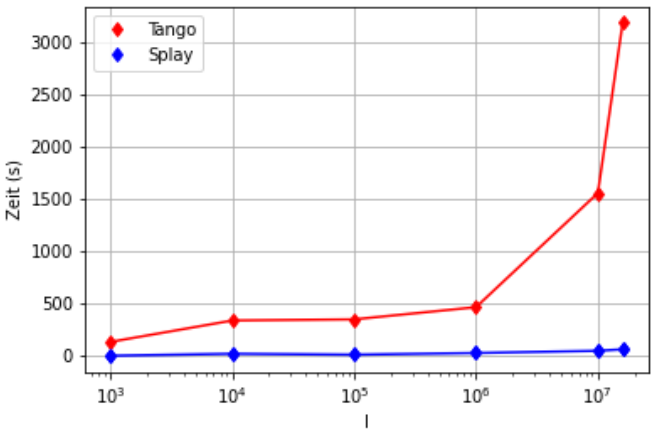

Abbildung 7: Laufzeittest mit zufälliger Zugriffsfolge.

n	Zeit Tango in (s)	Zeit Splay in (s)
10^3	138	6
10^4	342	16
10^5	353	16
10^6	469	32
10^7	1557	53
$1,6 * 10^7$	3181	67

Tabelle 1: Laufzeittest mit zufälliger Zugriffsfolge

1.2.2 Bit reversal permutation

Auf der X-Achse wird die Länge der einzelnen Binärdarstellungen l dargestellt. In der Spalte 2^l kann die Länge der Zugriffsfolge abgelesen werden. Die Ergebnisse bestätigen, dass es sich um eine aufwändige Zugriffsfolge für BST handelt.

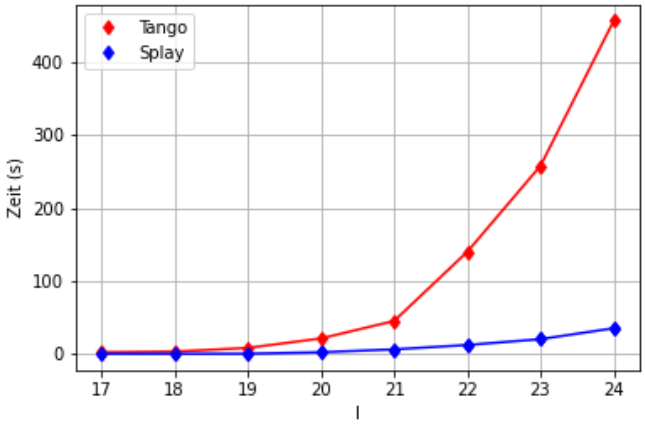

Abbildung 8: Laufzeittest bit reversal permutation.

l	2^l	Zeit Tango in (s)	Zeit Splay in (s)
17	131.072	2	1
18	262.144	3	1
19	524.288	8	1
20	1.048.576	21	2
21	2.097.152	45	6
22	4.194.304	140	12
23	8.388.608	258	20
24	16.777.216	457	35

Tabelle 2: Laufzeittest 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. Es wird dann nochmals die Anzahl von Zugriffen auf a hinzugefügt, um die gewünschte Länge der Zugriffsfolge zu erreichen. Die Anordnung der Schlüssel in der Zugriffsfolge geschieht wieder über einen Pseudozufallsgenerator.

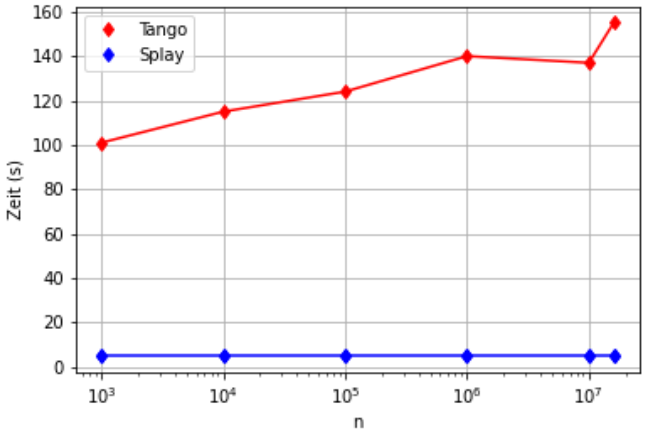

Abbildung 9: Laufzeittest static finger.

n	Zeit Tango in (s)	Zeit Splay in (s)
10^3	101	5
10^4	115	5
10^5	124	5
10^6	140	5
10^7	137	5
$1,6 * 10^7$	155	5

Tabelle 3: Laufzeittest static finger

1.2.4 Dynamic Finger

Beim ersten Test wird Zugriffsfolge $1, 3, 5, .., n-1, 1, 3, 5, .., n-1, ...$ verwendet. Der Abstand a zwischen zwei aufeinanderfolgenden Schlüssel ist also 2.

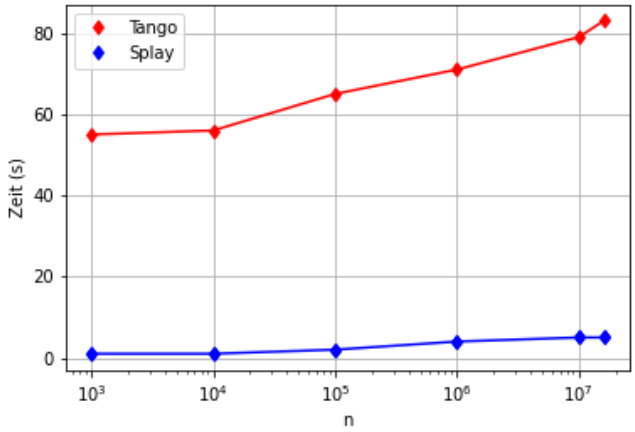

Abbildung 10: Laufzeittest dynamic finger.

n	Zeit Tango in (s)	Zeit Splay in (s)
10^3	55	1
10^4	56	1
10^5	65	2
10^6	71	4
10^7	79	5
$1,6 * 10^7$	83	5

Tabelle 4: Laufzeittest dynamic finger

Nun ist $n = 10.000.000$ fest, und es werden unterschiedliche Werte für a verwendet.

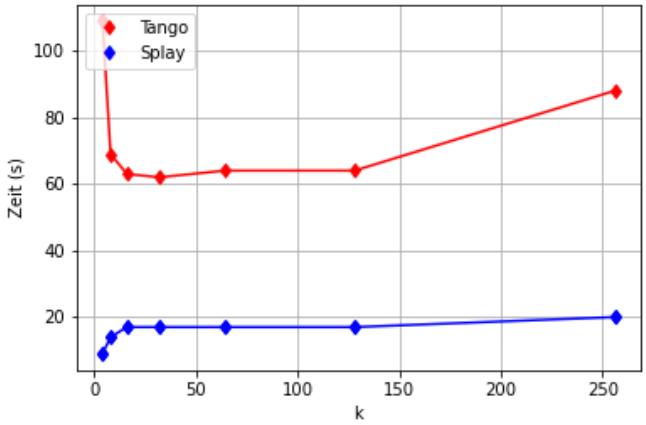

Abbildung 11: Laufzeittest dynamic finger, mit unterschiedlichen a.

a	Zeit Tango in (s)	Zeit Splay in (s)
4	109	9
8	69	14
16	63	17
32	62	17
64	64	17
128	64	17
256	88	20

Tabelle 5: Laufzeittest dynamic finger, mit unterschiedlichen a

1.2.5 Working Set

Das working set enthält 10 Prozent der Schlüssel des BST. Diese wurden zufällig ausgewählt. Die Zugriffsfolge besteht nur aus Schlüsseln des working set. Die Auswahl daraus geschieht dann für jede *access* Operation wieder zufällig.

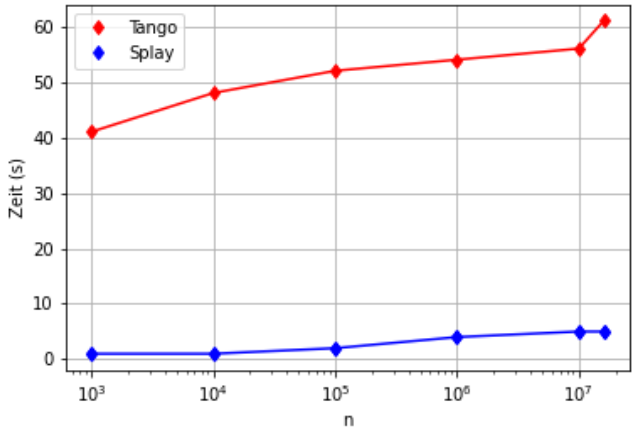

Abbildung 12: Laufzeittest working set.

n	Zeit Tango in (s)	Zeit Splay in (s)
10^3	101	5
10^4	115	5
10^5	124	5
10^6	140	5
10^7	137	5
$1,6 * 10^7$	155	5

Tabelle 6: Laufzeittest working set

Hier ist nun wieder $n = 10.000.000$ fest und der Prozentwert für die Anzahl der Schlüssel im working set veränderlich.

1.2.6 Weitere Laufzeittests

Alternierend die Schlüssel 1 und n:

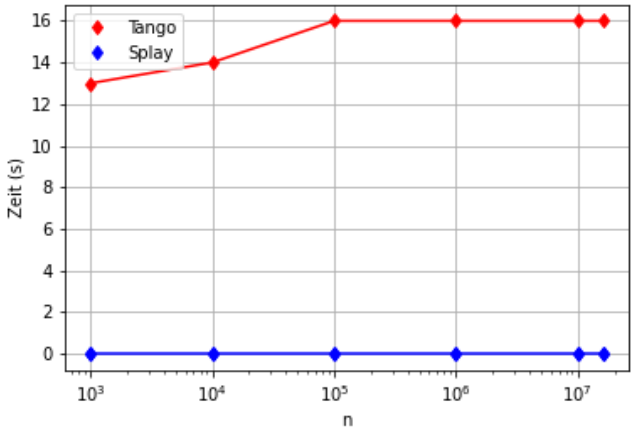

Abbildung 13: Laufzeittest alternierend.

n	Zeit Tango in (s)	Zeit Splay in (s)
10^3	13	1
10^4	14	1
10^5	16	1
10^6	16	1
10^7	16	1
$1,6 * 10^7$	16	1

Tabelle 7: Laufzeittest alternierend

Aufsteigend sortiert, mit insgesamt n Zugriffen:

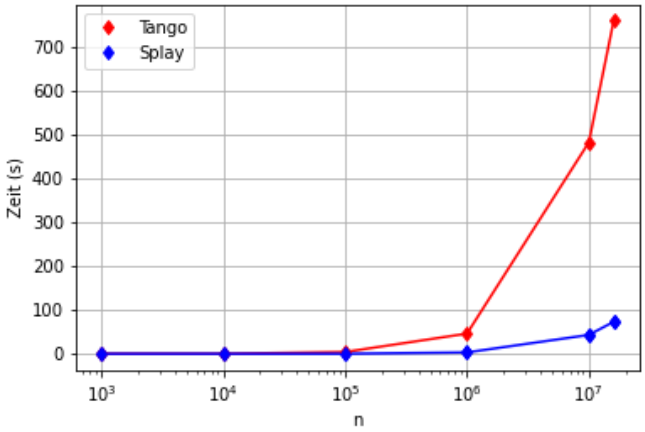

Abbildung 14: Laufzeittest aufsteigend sortiert.

n	Zeit Tango in (s)	Zeit Splay in (s)
10^3	1	1
10^4	1	1
10^5	4	1
10^6	46	3
10^7	483	43
$1,6 * 10^7$	761	73

Tabelle 8: Laufzeittest aufsteigend sortiert

Absteigend sortiert, mit insgesamt n Zugriffen:

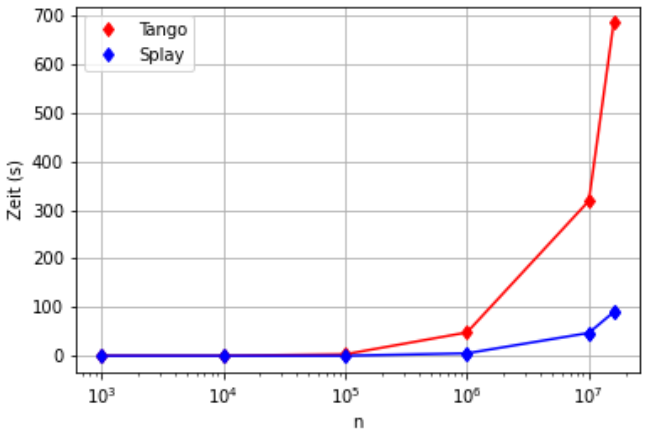

Abbildung 15: Laufzeittest absteigend sortiert.

n	Zeit Tango in (s)	Zeit Splay in (s)
10^3	1	1
10^4	1	1
10^5	3	1
10^6	48	5
10^7	319	47
$1,6 * 10^7$	686	90

Tabelle 9: Laufzeittest absteigend sortiert

Aufsteigend und absteigend sortiert, geschachtelt. Die Zugriffsfolge ist $1, n, 2, n-1, ..., n-2, 2, n-1, 1$

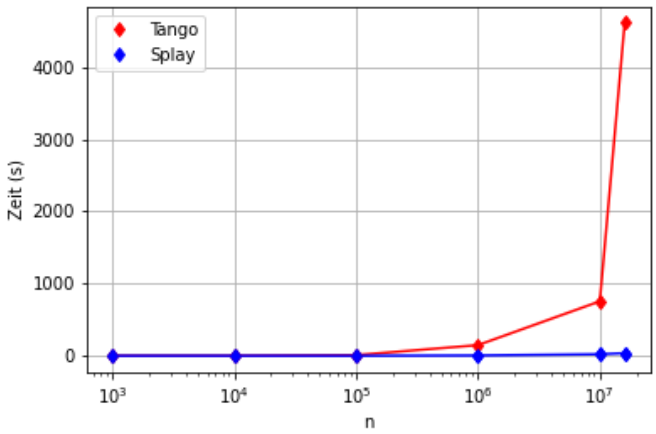

Abbildung 16: Laufzeittest auf- bzw. absteigend geschachtelt.

n	Zeit Tango in (s)	Zeit Splay in (s)
10^3	1	1
10^4	1	1
10^5	9	1
10^6	147	4
10^7	754	21
$1,6 * 10^7$	4611	36

Tabelle 10: Laufzeittest auf- bzw. absteigend geschachtelt.

2 Fazit und Ausblick

Der Tango Baum hat in den Praxistests im Vergleich zum Splay Baum wenig überzeugend abgeschnitten. Auch der Aufwand zur Implementierung ist für einen BST recht hoch. Bei ihm wurde die Idee mit den preferred child jedoch als Erstes umgesetzt, und hier könnten noch weitere interessante Variationen, mit zusätzlichen guten Laufzeiteigenschaften, folgen.
Mittlerweile hat sich zum Thema „dynamische Optimalität“ viel Literatur angesammelt. Hier wurde nur auf einen sehr kleinen Ausschnitt eingegangen. Zum Beispiel gibt es eine weitere untere Schranken für die Ausführungszeit von BST. Außerdem gibt es eine interessante geometrische Sicht auf BST, von der Aussagen über BST und andere Verfahren abgeleitet werden konnten. Es gibt auch deutlich mehr obere Laufzeitschranken, für Zugriffsfolgen

mit speziellen Eigenschaften, als sie hier vorgestellt wurden.
Ob es irgendwann gelingen wird die dynamische Optimalität, des Splay Baum oder irgendeiner anderen Variante eines BST zu beweisen, ist offen. Für den Einsatz bei gewöhnlich großen Schlüsselmengen, müssen dann aber auch die Summanden und Faktoren berücksichtigt werden, die in der O Notation vernachlässigt werden.

Literatur