Bachelorarbeit

Andreas Windorfer

8. Juli 2020

Zusammenfassung

Inhaltsverzeichnis

1 Zipper Baum

Der Zipper-Baum basiert auf dem Tango-Baum und nutzt auch preferred paths aus einem lower-bound-tree P. Aufbau und Pflege der preferred paths in P unterscheiden sich nicht vom Tango-Baum, wohl aber ihre Repräsentation im eigentlichen BST T. Der Zipper-Baum wurde in [1] vorgestellt. Er ist ebenfalls $\log(\log(n))$-competitive, garantiert aber auch $O(\log(n))$ im worst case, bei einer einzelnen *access* Operation. n steht wieder für de Anzahl der Knoten von T. Das Verhalten der Operationen *cut* und *concatenate* unterscheidet sich deutlich. Da die Operationen des Zipper Baumes recht aufwendig sind, werden sie hier wie in [1] in zwei Schritten vorgestellt. Zunächst an einem Hybrid Baum, der kein BST ist und dann am eigentlichen Zipper Baum. Die Knoten im Zipper und im Hybrid Baum sind genau so erweitert, wie die des Tango Baum. Sei v ein Knoten in P, dann ist in diesem Kapitel v^* der Knoten im Hybrid Baum bzw. Zipper Baum T, mit $key(v) = key(v^*)$.

1.1 Hybrid Baum

Znächst wird die Repräsentation eine preferred path in T vorgestellt. Dann wird die *access* Operationen, mit ihren Hilfsoperationen vorgestellt. Zum Schluss geht es noch um die Laufzeit.

1.1.1 Repräsentation eines preferred path

Die Repräsentation eines preferred path $P_p = p_1, p_2, .., p_p$ in T stellt einen Hilfsbaum H dar, der in zwei Teile unterteilt wird, den **top path** und dem **bottom tree**. Der bottom tree ist ein balancierter BST der genau die Schlüssel enthält, die in P_p enthalten sind jedoch nicht im top path. Der top path enthält $n_1 \in [\log(\log(n)), 3\log(\log(n))]$ Knoten falls ein bottom tree existiert, ansonsten $n_1 \in [1, 3\log(\log(n))]$. Der top path besteht aus den Knoten ${p_1}^*, {p_2}^*, .., {p_{n_1}}^*$ und stellt eine eins zu eins Repräsentation von $p_1, p_2, .., p_{n_1}$ dar. ${p_1}^*$ ist die Wurzel von H. Die Wurzel des bottom tree ist ein Kind von ${p_{n_1}}^*$. Abbildung 2 zeigt eine mögliche Darstellung zu dem preferred path in Abbildung 1. Jeder Knoten in H kann in $O(\log(\log(n)))$ Zeit erreicht werden. Die Repräsentationen der preferred paths werden wie beim Tango Baum zu einer Gesamtstruktur entsprechend ihrer Schlüssel zusammengefügt.

1.1.2 Die *access* Operation beim Hybrid Baum

Sei p der Zeiger der $access(k)$ Operation in T. Erreicht p während des Suchens in einem Hilfsbaum H_1 die Wurzel eines weiteren Hilfsbaum H_2, werden die

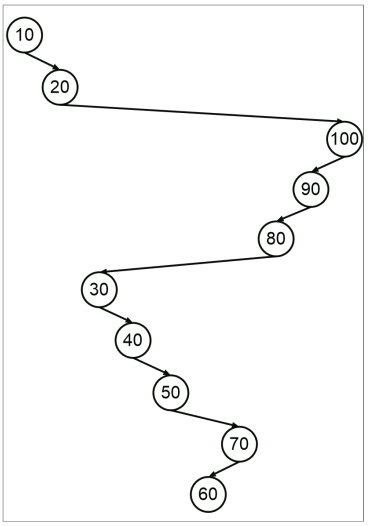

Abbildung 1: Beispiel preferred path

Knoten von H_1, entsprechend ihrer *depth* Variable, vergleichbar mit dem Tango Baum, mit einer *cut* Operation in zwei Bäume D_1 und D_2 aufgeteilt. Sei D_2 der Baum mit den Knoten größerer Tiefe. D_2 wird sofort in eine gültige Pfadrepräsentation überführt, wie genau wird in *cut* beschrieben. Die Knoten in D_1 werden nach der Ausführung von *access* im Hilfsbaum mit der Wurzel von T enthalten sein. Dieser wird mit *concatenate* erst erstellt wenn k gefunden ist. D_1 bleibt also zunächst unverändert bestehen.

***cut* beim Hybrid Baum** Es werden zwei Fälle unterschieden. Beim ersten Fall zeigte p in H_1 nur auf Knoten im top path $q_1, q_2, .., q_m$ im zweiten Fall erreichte p den bottom tree. Es wird davon ausgegangen dass in H_1 ein bottom tree enthalten ist, der andere Fall ist einfacher und ergibt sich aus der Beschreibung.
Fall 1:
Sei q_i der Vater von H_2, dann muss eine Laufzeit von $O(i)$ erreicht werden. q_{i+1} wird zur Wurzel von H_2. Es sichergestellt werden, dass die Anzahl der Knoten im top path von H_2 groß genug ist. Enthält er weniger als $2\log(\log(n))$ Knoten, werden, mit einem **Extraktionsprozess**, Knoten aus dem bottom tree dem top path hinzugefügt. Dazu werden *extract* Operationen verwendet. Eine solche Operation fügt dem top path $\log(\log(n))$ Knoten aus dem bottom tree hinzu. Um die Laufzeit einhalten zu können verteilt textitextract dies über mehrere *cut* Operationen. Dazu wird der bottom tree in einen Zwischenzustand versetzt, der keinem balancierten BST entspricht. Ein Hilfsbaum bei dem kein Extraktionsprozess aktiv ist hat $2\log(\log(n))$

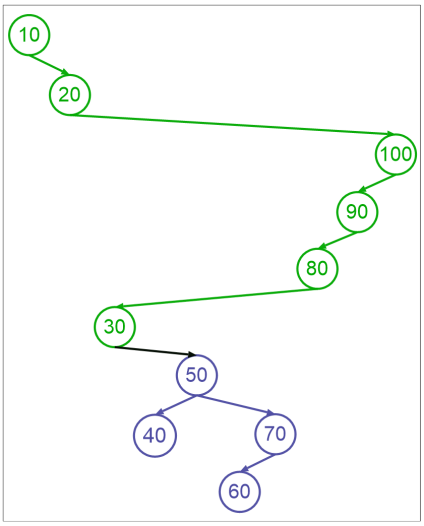

Abbildung 2: Mögliche Repräsentation des preferred path aus 1. Grün der top path, blau der bottom tree.

Knoten im top path. Der Extraktionsprozess wird im Abschnitt zu *extract* beschrieben. Fall 2:
Es muss eine Laufzeit von $O\log(\log(n))$ erreicht werden.Dieser Fall ist sehr ähnlich zum Tango Baum und es werden die Operationen von ihm verwendet. Sei *tangoConcatenate* die *concatenate* Operation wie im Kapitel zum Tango Baum beschrieben und *tangoCut* die *cut* Operation von dort. Sei v der Vater der Wurzel von H_2 in T. Ist in H_1 ein Extraktionsprozess aktiv, muss dieser beendet werden und der top path muss in einen balancierten Baum H_3 überführt werden. Nun werden H_1 und H_3 mit *tangoConcatenate* zu H_5 vereinigt. Dann wird *tangoCut* verwendet um die Knoten mit einem Wert der *depth* Variable größer v. *depth* aus in einen Hilfsbaum H_6 auszulagern. Auf H_6 werden zwei Extraktionsprozess ausgeführt um eine gültige Pfadrepräsentation zu erreichen.
Im ersten Fall entstehen Kosten von $O(i)$ um q_i zu erreichen, und dass auch in *extract* Kosten von $O(i)$ entstehen wird weiter unten gezeigt. Im zweiten Fall entstehen Kosten von $O(i)$ sowohl um v zu erreichen, als auch in *extract*.

***concatenate* beim Hybrid Baum** *concatenate* wird verwendet um zum Abschluss von *access* den Hilfsbaum H mit der Wurzel von T zu erstellen. Seien $H_1, H_2, .., H_m$ die Bäume die zu H vereinigt werden, so dass für $i \in \{1, 2, ..m\}$ und $i > 1$, in H_{i-1} der Vater der Wurzel von H_i enthalten ist. Abbildung 3 zeigt eine beispielhafte Konstellation. Zunächst werden die Bäume $H_1, H_2, .., H_m$ zu balancierten Bäumen $H'_1, H'_2, .., H'_m$ transformiert.

Diese werden dann mit einer Folge von $m-1$ *tangoConcatenate* Operationen zu H zusammengefügt. Als erstes werden H'_1 und H'_2 zusammengefügt. Zu dem neu entstandenen Hilfsbaum wird dann H'_3 hinzugefügt usw. Um H zu erstellen werden dann noch $2\log(\log(n))$ Knoten zu einem top path extrahiert. Abbildung 4 zeigt den Ablauf für das Beispiel.
Um H_i zu H'_i zu überführen werden die Knoten in seinem top path als Bäume mit einem Knoten betrachtet. Diese werden mit Folgen von *tangoConcatenate* zusammengefügt. In jeder solchen Folge wird die Anzahl der Bäume halbiert, indem zwei aufeinanderfolgende Bäume jeweils zusammengeführt werden. Nach der Ausführung dieser *tangoConcatenate* Folgen existiert ein aus dem top path entstandener balancierter BST und eventuell ein bottom tree. Diese werden mit einer weiteren *tangoConcatenate* Operation zusammengeführt. Nach jeder *tangoConcatenate* Folge halbiert sich die Anzahl der aus dem top path entstandenen Bäume und die Anzahl ihrer Knoten verdoppelt sich. Sei n_i die Anzahl der Knoten von H_i, dann entstehen pro Folge Kosten von $O(\log n_i)$ und es werden $O(\log n_i)$ Folgen benötigt. Aufaddieren über alle i ergibt damit $O(\log(\log n))$.
tangoConcatenate hat eine Laufzeit von $O(\log(n_r))$, mit n_r ist die Anzahl der Knoten, des resultierenden Baumes. Die Anzahl der benötigten Operationen is $O(\log(n))$. Damit ergibt sich auch für die Gesamtlaufzeit $O(\log(\log n))$.

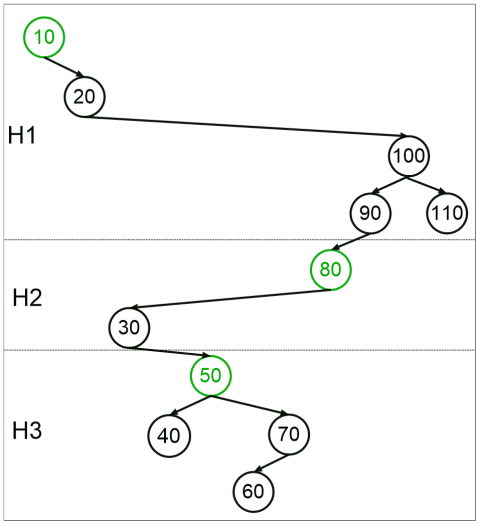

Abbildung 3: Mögliche Ausgangsituation vor *concatenate*.

***extract* beim Hybrid Baum** Wie bereits erwähnt wird diese Operation benötigt um einen Extraktionsprozess durchzuführen. Sei H eine Repräsentation des preferred path $P_p = p_1, p_2, .., p_m$ und $i \in \{1, 2, .., m\}$. Sei

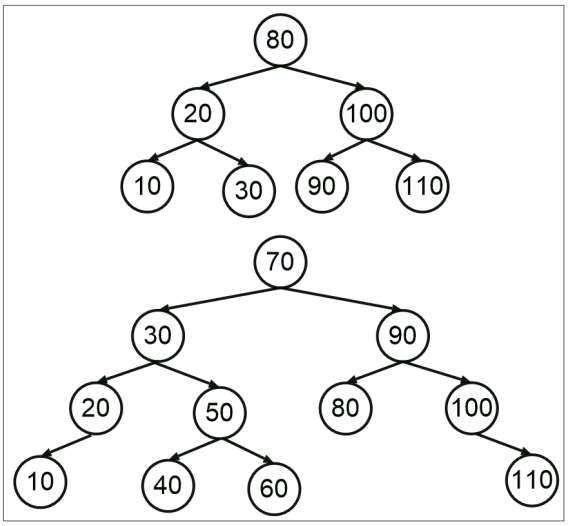

Abbildung 4: Es werden zwei *concatenate* Operationen benötigt. Oben das Ergebnis der Ersten, unten dass der Zweiten

$n_1 = \log(\log(n))$. Sind genügend Knoten im bottom tree vorhanden, bereitet *extract* immer die Extraktion von n_1 Knoten vor. Existieren noch genügend vorbereitete Knoten kann ein verkürzter Ablauf erfolgen. Ansonsten werden die vorbereiteten Knoten extrahiert bevor ein neuer Extraktionsprozess begonnen wird. Sei v der Vater der Wurzel des bottom tree, mit $d = v.depth$ und p_i der Knoten in P_p mit Schlüssel $key(v)$. Werden n_2 Knoten zum extrahieren vorbereitet, dann müssen dies der Reihe nach die Knoten mit *depth* Wert $d+1, d+2, .., d+n_2$ sein, damit ein valider top path entsteht. P_p wird in **zig Segmente** und **zag Segmente** unterteilt. zig Segmente entsprechen den längst möglichen Teilpfaden von Knoten mit linken Kindern in P_p. zag Segmente entsprechen den längst möglichen Teilpfaden von Knoten mit rechten Kindern in P_p. Dass Blatt P_p wird dem Segment seines Vaters zugeordnet. In Abbildung 7 sind zig und zag Segmente dargestellt. Sei S_{zig} die Folge der Knoten der zig Segmente, aufsteigend sortiert nach der Tiefe und S_{zag} die Folge der Knoten der zag Segmente, aufsteigend sortiert nach der Tiefe der enthaltenen Knoten. Ist u der tiefste Knoten eines zig bzw. zag Segmentes, so können in den Segmenten mit Knoten größerer Tiefe nur noch größere bzw. kleinere Schlüssel enthalten sein, vergleiche Abschnitt **??** und Abbildung 7. Deshalb müssen die Knoten in $S_1 \circ S_2$ aufsteigend sortiert nach Schlüssel sein. Da die Knoten in S_{zig} bzw. S_{zag} aber auch aufsteigend bzw. absteigend nach Tiefe sortiert sind, können aus $S_{zig} \circ S_{zag}$ Intervalle $[l, r]$ für die Schlüsselmengen, von Pfaden in P_p mit Endknoten p_m abgeleitet werden, siehe Abbildung 7, wobei l in S_{zig} enthalten sein muss und r in S_{zag}. Sei $[l, r]$

ein so erstelltes Intervall zu dem Pfad von p_{i+n_1} zu p_m. Es müssen genau die Knoten im bottom tree zum extrahieren vorbereitet werden, deren Schlüssel nicht in $[l, r]$ liegen. Die Knoten mit Schlüssel l und r, sowie der Vorgänger v_l von l mit Schlüssel l' und der Nachfolger v_r von r mit Schlüssel r' werden gefunden wie in Abschnitt **??** gezeigt. Es wird eine *split* Operation wie beim Kapitel über den Tango Baum verwendet, die der bottom tree zur Verfügung stellen muss. Mit zwei solcher Operationen werden die Knoten v_l und v_r herausgelöst. Das linke Kind von v_l ist ein balancierter BST B, der alle Schlüssel aus H enthält die kleiner als l sind. Das rechte Kind von v_l ist v_r. Das linke Kind von v_r ist ein balancierter BST D, der alle Schlüssel aus H enthält die in (l, r) enthalten sind. Das rechte Kind von v_r ist ein balancierter BST E, der alle Schlüssel aus H enthält die größer als r sind. Es werden also genau die Knoten aus B und E vorbereitet. Schritt drei in Abbildung 5 ist damit bereits vollzogen. In Schritt drei werden B und E nun zu Listen gewandelt. Sei B' bzw. E' der Baum der durch die Umwandlung von B bzw. E zu einer Liste entsteht. Es wird B' betrachtet, E' ist dazu symmetrisch. Die Wurzel von B' enthält den größten Schlüssel in B', somit sind alle anderen Knoten in B' linke Kinder. Die Umwandlung geschieht wie folgt: Es werden so oft Links Rotationen auf das rechte Kind der Wurzel ausgeführt, bis der rechte Teilbaum der Wurzel leer ist. In Schritt zwei wird der rechte Teilbaum des linken Kindes der Wurzel auf die gleiche Art und Weise geleert und eine Rechts Rotation auf das linke Kind der Wurzel ausgeführt. Schritt zwei wird so oft wiederholt bis B' erzeugt ist. Abbildung 6 stellt diesen Vorgang dar. Da l in S_{zig} enthalten ist, müssen alle Schlüssel aus B' auch als Schlüssel von Knoten S_{zig} enthalten sein. S_{zig} ist aufsteigend nach den Schlüsseln als auch nach den Tiefen sortiert, somit müssen dies auch die Knoten in B' sein. Das bedeutet für einen Knoten v aus B', dass alle Knoten in seinem rechten Teilbaum einen größeren Wert bei der *depth* Variable haben als $v.depth$. Für v aus E' gilt für den linken Teilbaum das gleiche. Aus den gleichen Gründen muss $v_l.depth$ bzw. $v_r.depth$ einen größeren Wert haben, als die *depth* Variablen aller Knoten in B' bzw. E'. v_l, v_r sowie die Knoten aus B' und E' sind nun zum extrahieren vorbereitet. Sei u die Wurzel von B' oder E', je nachdem bei welchem Knoten die *depth* Variable einen kleineren Wert hat. Dann kann u mit maximal zwei Rotationen den top path hinzugefügt werden. Sind B' und E' erschöpft und auch v_r und v_l dem top path hinzugefügt, beginnt der Vorgang von vorne. Muss ein Extraktionsprozess aufgrund *cut* vorzeitig abgebrochen werden, werden aus B' und E' wieder balancierte BST erzeugt, simultan zu der Beschreibung in *concatenate*. Im Anschluss wird mit zwei *tangoConcatenate* Operationen wieder ein bottom tree erstellt. Um die Laufzeit von *cut* einhalten zu können, werden die Knoten im Hybrid Tree um einen weiteren Zeiger erweitert. Ist in einem Hilfsbaum H mit Wurzel w ein

Extraktionsprozess aktiv, zeigt dieser auf den Knoten der als nächstes extrahiert wird. Da p nach dem extrahieren von Knoten, nochmals auf w zeigen muss, bevor er auf einen Knoten außerhalb von H zeigt, kann dieser Knoten einfach gepflegt werden. Aufgrund dieses Zeigers ist der Hybrid Baum jedoch kein BST.
Bei Schritt eins und zwei entstehen Kosten von $O(\log(\log n))$, vergleichbar mit Kapitel **??**. Beim erstellen von B' aus B ist jeder Knoten an maximal drei Rotationen beteiligt, vergleiche Abbildung 6. Da maximal $\log(\log n)$ Knoten vorbereitet werden, entstehen Kosten von $O(\log(\log n))$. Bei Schritt vier kommen noch einmal Kosten von $O(\log(\log n))$ hinzu. Werden mit einem verkürzten Ablauf k bereits vorbereitete Knoten dem top path hinzugefügt, entstehen mit Hilfe des zusätzlichen Zeigers Kosten von $O(k)$.

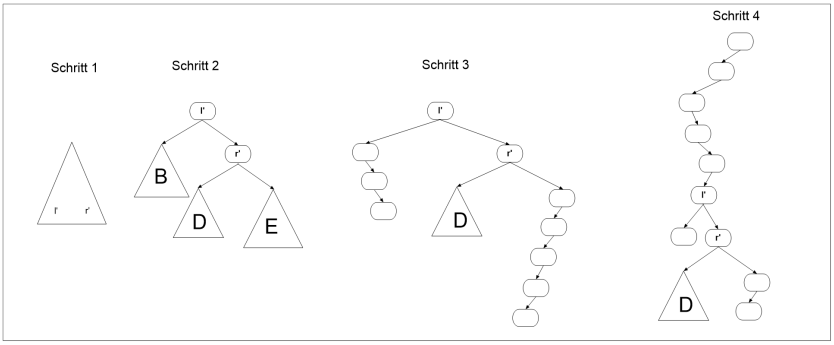

Abbildung 5: Es wird ein Extraktionsprozess dargestellt. Die Abbildung basiert auf einer in [1]

1.1.3 Laufzeitanalyse beim Hybrid Baum

Zunächst wird ein Lemma benötigt.

Lemma 1.1. *Sei T ein Hybrid Baum mit n Knoten und Y ein dazu erstellter lower bound tree. Sei v ein Knoten aus T und u der Knoten in Y mit $key(v) = key(u)$, dann gilt $depth(v) = O(depth(u))$.*

Beweis. Seien $P_1, P_2, .., P_m$ die preferred path die Knoten aus dem Pfad P_u von der Wurzel von Y zu u enthalten. Jeder von ihnen wird nun als BST betrachtet. Für $i \in \{1, 2, .., m\}$ sei H_i die Pfadrepräsentation zu P_i. Sei p_i der Knoten in P_i mit der größten Tiefe unter denen aus P_u. (Es muss $p_m = u$ gelten) Sei h_i der Knoten in H_i mit $key(h_i) = key(y_i)$. Die folgenden Tiefenangaben werden auf P_i bzw. H_i bezogen. Ist h_i im top path enthalten, haben h_i und p_i die gleiche Tiefe. Liegt h_i im bottom tree so muss

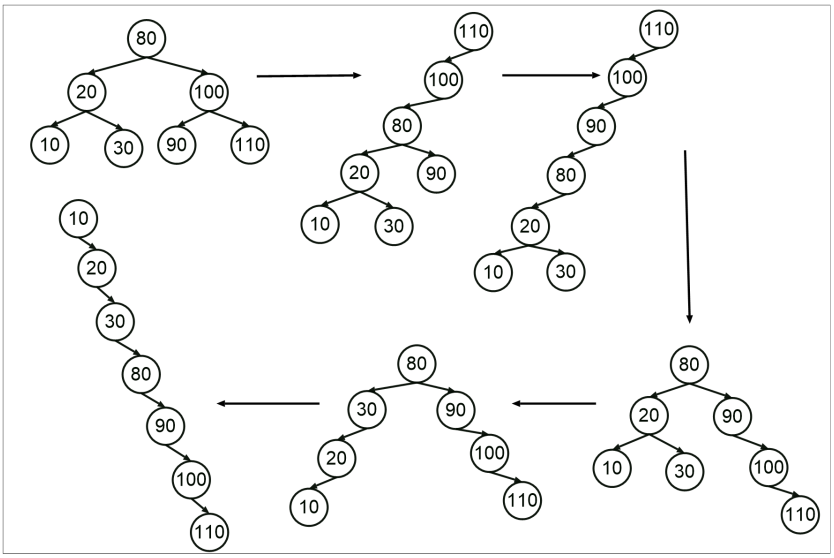

Abbildung 6: Beispielhaftes überführen von B zu B'.

$depth\,(p_i) > \log(\log n)$ gelten aufgrund der Konstruktion von H_i. Für die Tiefe jedes Knoten in H_i gilt $O\,(\log(\log n))$

□

Daraus ergibt sich sofort für die Höhe eines Hybrid Baum mit n Knoten sofort $O\,(\log n)$.

1.2 Repräsentation eines preferred path

Ein Hilfsbaum H zur Repräsentation eines preffered path $P_p = p_1, p_2, .., p_p$ wird in einen **zipper** und einem **bottom tree** unterteilt. Enthält der preferred path nicht mehr als $O \log(\log(n))$ Knoten, besteht der Hilfsbaum allein aus dem zipper. Die Anzahl der Knoten des zipper liegt in
$[\log(\log(n))/2, 2\log(\log(n))]$, wenn ein bottom tree existiert, ansonsten in $[0, 2\log(\log(n))]$. Der bottom tree ist ein balancierter BST, der genau die Schlüssel aus P_p enthält, die im zipper fehlen. Enthält der zipper q Knoten, dann entsprechen deren Schlüsseln denen aus dem Pfad $p_1, p_2, .., p_q$. Der zipper ist ein BST und die Wurzel des bottom tree ist das Kind eines Knotens aus dem zipper, so dass H ein BST ist.
Die Konstruktion des zipper ist so ausgelegt, dass innerhalb konstanter Zeit von der Wurzel von H auf die Wurzel des bottom tree zugegriffen werden kann. Es gibt in der Regel mehrere mögliche Darstellungen eines zipper zu P_p. Ein zipper z besteht ebenfalls wieder aus zwei Bestandteilen dem **oberen zipper** z_1 und dem **unterem zipper** z_2. Insgesamt müssen in z zumindest

$\log(\log(n))/2$ enthalten sein. Pro Bestandteil dürfen maximal $\log(\log(n))$ Knoten enthalten sein. Sei a_1 die Anzahl der Knoten in z_1 und a_2 die in z_2, so dass die genannten Anforderungen eingehalten werden. Konstellationen in denen das nicht möglich ist, bleiben zunächst außen vor. In z_1 sind die Schlüssel der Knoten des Pfades $P_1 = p_1, p_2, .., p_{a_1}$ enthalten. In z_2 die aus $P_2 = p_{a_1+1}, p_{a_1+2}, .., p_{a_1+a_2}$.
P_1 und P_2 werden in **zig Segmente** und **zag Segmente** unterteilt. zig Segmente entsprechen den längst möglichen Teilpfaden von Knoten mit linken Kindern in P_p. zag Segmente entsprechen den längst möglichen Teilpfaden von Knoten mit rechten Kindern in P_p. Enthält P_2 das Blatt aus P_p, wird dieses dem Segment seines Vaters zugeordnet. In Abbildung 7 sind zig und zag Segmente dargestellt. Sei S_{zig} die Folge der Knoten der zig Segmente, aufsteigend sortiert nach der Tiefe und $_{zag}$ die Folge der Knoten der zag Segmente, aufsteigend sortiert nach der Tiefe der enthaltenen Knoten. Ist u der tiefste Knoten eines zig bzw. zag Segmentes, so können in den Segmenten mit Knoten größerer Tiefe nur noch größere bzw. kleinere Schlüssel enthalten sein, vergleiche Abschnitt **??** und Abbildung 7. Deshalb müssen die Knoten in $S_1 \circ S_2$ aufsteigend sortiert nach Schlüssel sein. Da die Knoten in S_{zig} bzw. S_{zag} aber auch aufsteigend bzw. absteigend nach Tiefe sortiert sind, können aus $S_{zig} \circ S_{zag}$ Intervalle für die Schlüsselmengen, von Pfaden in P_p mit Endknoten p_p abgeleitet werden, siehe Abbildung 7.
Eine Folge von Knoten S_l^r entstehe aus einer Folge S durch entfernen aller Knoten mit einer Tiefe außerhalb von $[l, r]$. Sei l_1 der Knoten in ${S_{zig}}_0^{a_1}$ mit der größten Tiefe, und r_1 der Knoten in ${S_{zag}}_{a_1}^{a_2}$ mit der größten Tiefe. ${l_1}^*$ ist die Wurzel von H. Der linke Teilbaum von ${l_1}^*$ enthält ausschließlich die Knoten ${p_1}^*, {p_2}^*, .., {p_{a_1}}^*$, so dass für $i \in \{2, 3, .., a_1\}$ gilt, ${p_{i-1}}^*$ ist das rechte Kind von ${p_i}^*$. r^* ist das rechte Kind von ${l_1}^*$ und sein rechter Teilbaum enthält ausschließlich die Knoten ${p_{a_1+1}}^*, {p_{a_1+2}}^*, .., {p_{a_1+a_2}}^*$, so dass für $i \in \{a_1 + 2, a_1 + 3, .., a_2\}$ gilt, ${p_{i-1}}^*$ ist das linke Kind von p_i*. Abbildung **??** zeigt einen so erstellten oberen Teil eines zipper. z_2 wird simultan aus S_{zig}, S_{zag} und $[a_1 + 1, a_2]$, mit Knoten l_2 und r_2 erzeugt. Die Wurzel ${l_2}^*$ von z_2 ist das linke Kind von ${r_1}^*$. Um die Links-Rechts-Beziehung einzuhalten muss die Wurzel eines vorhandenem bottom tree das linke Kind von ${r_2}^*$ sein. Abbildung **??** zeigt eine mögliche Darstellung von z. Genau wie beim Tango-Baum werden die Hilfsbäume zu den preferred path zu einem gesamten BST, den Zipper-Baum, zusammengefügt.

1.3 Die *access* Operation beim Zipper Baum

Wie der Tango Baum erweitert der Zipper Baum seine Knoten um eine *isRoot* Variable sowie um die *depth*, *minDepth* und *maxDepth* Variable. Sei k der Parameter der Operation und p der Zeiger der Operation in den BST. Diese *access* Operation hat Ähnlichkeit zu der des Tango Baum. Nach der Operation muss es einen preferred path von der Wurzel von P zu dem Knoten mit Schlüssel k geben. Verlässt p einen Hilfsbaum H, werden aus H zwei BST H_1 und H_2 mit Schlüsselmengen analog zum Tango Baum erstellt. Sei H_1 der BST mit den Knoten mit kleinerer Tiefe und H_2 der mit den Knoten mit größerer Tiefe. H_2 wird sofort in einen gültigen Hilfsbaum überführt. Die Knoten aus H_1 sind nach der Operation in dem Hilfsbaum enthalten, der die Wurzel des Zipper Baumes enthält. Dieser wird erst erzeugt nachdem k gefunden wurde. Insbesondere startet p nicht mehrmals bei der Wurzel von T. Es werden wieder Hilfsoperationen benötigt.

zip Operation

access Operation Nun wird die *access* Operation des Tango Baumes betrachtet. Sei k der Parameter der Operation und p der Zeiger der Operation in den BST. Solange sich p im Hilfsbaum mit der Wurzel des Tango Baumes T befindet, verhält sich die Operation wie die Standardvariante von *search*. Erreicht p die Wurzel eines anderen Hilfsbaumes H_2, muss sich ein preferred child in P verändert haben. T wird mit *cut* und *join* so angepasst, das er wieder die preferred paths in P repräsentiert. Anschließend startet p wieder an der Wurzel von T. Erreicht p den Knoten mit *key* (k) so wird das preferred child des Knoten mit Schlüssel k in P auf *left* gesetzt. So dass nochmals eine Anpassung notwendig sein kann. Die Operation wird noch etwas detaillierter beschrieben.

1.4 Laufzeitanalyse für *access*

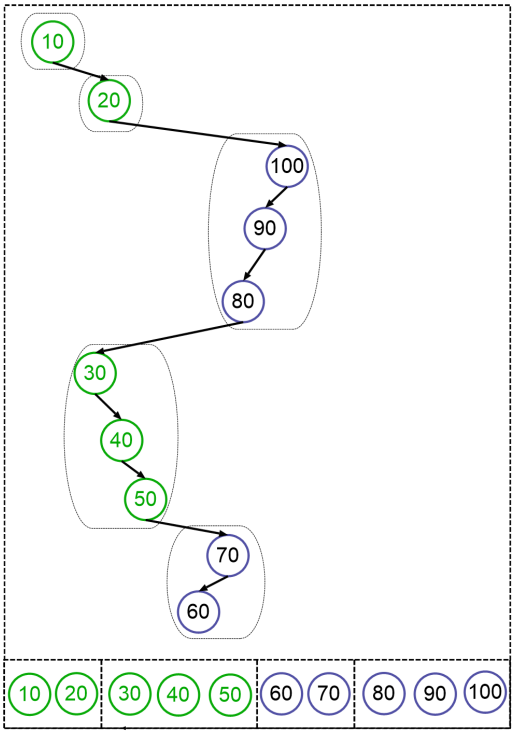

Abbildung 7: zig Segmente sind grün dargestellt. zag Segmente blau

Literatur

[1] Prosenjit Bose, Karim Douïeb, Vida Dujmović, and Rolf Fagerberg. An o(log log n)-competitive binary search tree with optimal worst-case access times. *Algorithm Theory - SWAT 2010*, page 38–49, 2010.