

# Bachelorarbeit

Andreas Windorfer

1. August 2020

## Zusammenfassung

# Inhaltsverzeichnis

<b>1</b>	<b>Zipper Baum</b>	<b>4</b>
1.1	Hybrid Baum . . . . .	4
1.1.1	Repräsentation eines preferred path beim Hybrid Baum	4
1.1.2	Die <i>access</i> Operation beim Hybrid Baum . . . . .	4
1.1.3	Laufzeitanalyse beim Hybrid Baum . . . . .	10
1.2	Repräsentation eines preferred path beim Zipper Baum . . . .	11
1.3	Die <i>access</i> Operation beim Zipper Baum . . . . .	12
1.4	Laufzeitanalyse beim Zipper Baum . . . . .	14

# 1 Zipper Baum

Der Zipper Baum basiert auf dem Tango Baum und nutzt auch preferred paths aus einem Referenzbaum  $P$ . 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 die Anzahl der Knoten von  $T$ . Das Verhalten der Operationen *cut* und *join* 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 Baum sind wie beim Tango Baum erweitert. Beim Hybrid Baum kommt ein zusätzlicher Zeiger auf einen Knoten hinzu. Sei  $v$  ein Knoten in  $T$ , dann ist in diesem Kapitel  $v^*$  der Knoten in  $P$ , mit  $key(v) = key(v^*)$ .

## 1.1 Hybrid Baum

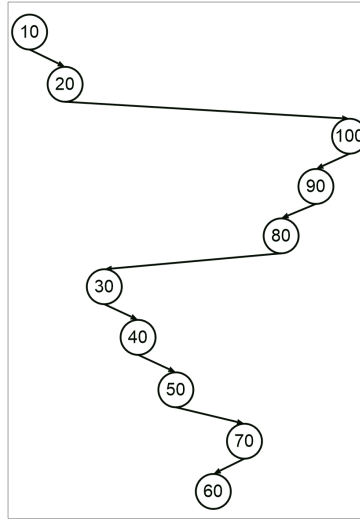
Znächst wird die Repräsentation eines preferred path in  $T$  vorgestellt. Dann wird die *access* Operationen, mit ihren Hilfsoperationen vorgestellt. Zum Schluss geht es noch um die Laufzeit.

### 1.1.1 Repräsentation eines preferred path beim Hybrid Baum

Die Repräsentation eines preferred path  $P_p = p_1^*, p_2^*, \dots, p_m^*$  in  $T$  stellt einen Hilfsbaum  $H$  dar, der in zwei Teile unterteilt wird, den **top path** und den **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  $p_1, p_2, \dots, p_{n_1}$  und stellt eine eins zu eins Repräsentation von  $p_1^*, p_2^*, \dots, p_{n_1}^*$  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



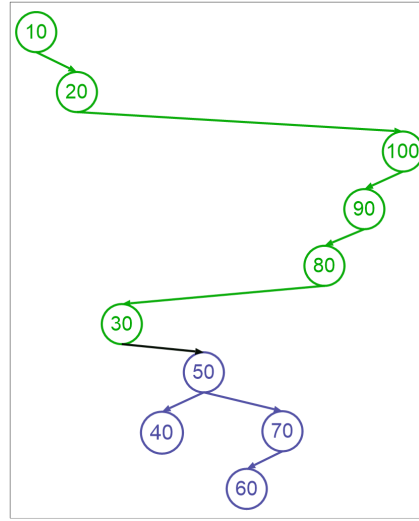
**Abbildung 1:** Beispiel preferred path

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, bei dem die Werte der *depth* Variablen die größeren Werte annehmen.  $D_2$  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 *join* erst erstellt, wenn der Knoten mit Schlüssel  $k$  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  $p_1, p_2, \dots, p_m$  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 Elternknoten der Wurzel von  $H_2$ , dann muss eine Laufzeit von  $O(i)$  erreicht werden.  $q_{i+1}$  wird zur Wurzel von  $H_1$ . Es muss sichergestellt werden, dass die Anzahl der Knoten im top path von  $H_1$  noch 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 wird eine *extract* Operation verwendet. Eine solche Operation fügt dem top path  $\log(\log(n))$  Knoten aus dem bottom tree hinzu. Um die Laufzeit einhalten zu können verteilt *extract* dies über mehrere *cut* Operationen. Dazu wird der bottom tree in einen Zwischenzustand versetzt, der keinem balan-



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

cierten BST entspricht. Ein Hilfsbaum bei dem kein Extraktionsprozess aktiv ist hat  $2 \log(\log(n))$  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 *tangoJoin* die *join* Operation wie im Kapitel zum Tango Baum beschrieben und *tangoCut* die *cut* Operation von dort. Sei  $v$  der Vater der Wurzel von  $H_2$ . 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  $H_3$  und der bottom tree mit *tangoJoin* zu  $H_4$  vereinigt. Dann wird *tangoCut* verwendet um die Knoten mit einem Wert der *depth* Variable größer  $v.depth$  aus in einen Hilfsbaum  $H_5$  auszulagern. Auf  $H_5$  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  $O(\log(\log(n)))$  sowohl um  $v$  zu erreichen, um  $H_4$  zu erzeugen (siehe *join*) als auch in *extract*.

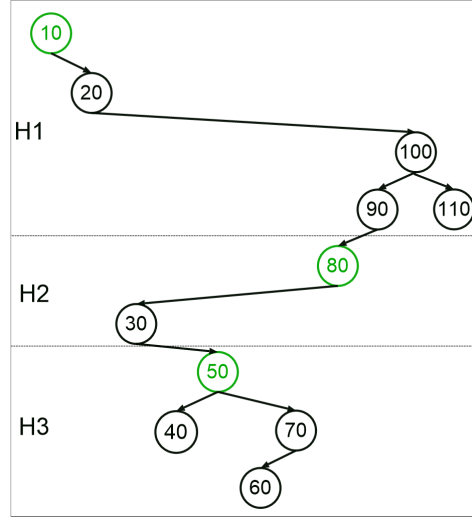
**join beim Hybrid Baum** *join* wird verwendet um zum Abschluss von *access* den Hilfsbaum  $H$  mit der Wurzel von  $T$  zu erstellen. Seien  $H_1, H_2, \dots, H_m$  die Bäume die zu  $H$  vereinigt werden, so dass für  $i \in \{1, 2, \dots, m\}$  und  $i > 1$ ,

in  $H_{i-1}$  ist der Vater der Wurzel von  $H_i$  enthalten gilt. Abbildung 3 zeigt eine beispielhafte Konstellation. Zunächst werden die Bäume  $H_1, H_2, \dots, H_m$  zu balancierten Bäumen  $H'_1, H'_2, \dots, H'_m$  transformiert. Diese werden dann mit einer Folge von  $m - 1$  *tangoJoin* 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 *tangoJoin* zusammengefügt. In jeder solchen Folge wird die Anzahl der Bäume halbiert, indem zwei aufeinanderfolgende Bäume jeweils zusammengeführt werden. Nach jeder *tangoJoin* 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 im top path von  $H_i$ , dann summieren sich die Kosten über alle Folgen zu  $O\left(\sum_{j=1}^{\log n_i} j n_i / 2^j\right) = O(n_i)$ . Diese Kosten sind bereits während des Suchens in  $H_i$  entstanden und können deshalb vernachlässigt werden. Nach der Ausführung dieser *tangoJoin* Folgen existiert ein aus dem top path entstandener balancierter BST und eventuell ein bottom tree. Diese werden mit einer weiteren *tangoJoin* Operation zusammengeführt. Die Kosten für diese Operation betragen  $O(\log(\log(n)))$  und sind bereits beim Suchen im bottom tree von  $H_i$  angefallen. Die Gesamtkosten zum erstellen von  $H_i$  können also asymptotisch betrachtet vernachlässigt werden.

Die Kosten um  $H$  aus  $H'_1, H'_2, \dots, H'_m$  zu bilden, sind analog zum Tango Baum. Für den Extraktionsprozess entstehen nochmals Kosten von  $O(\log(\log(n)))$ .

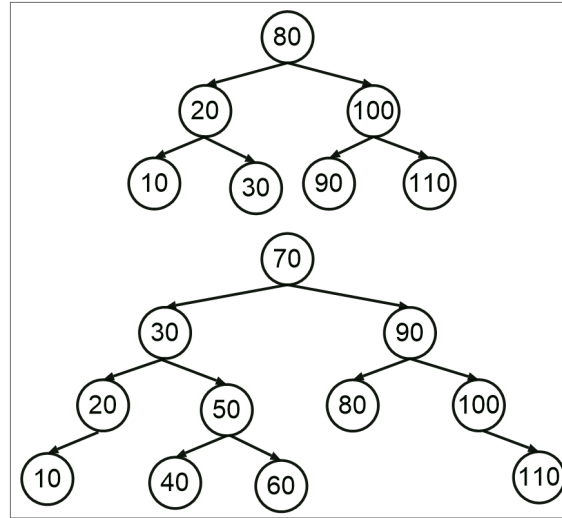
**extract beim Hybrid Baum** Wie bereits erwähnt wird diese Operation benötigt um einen Extraktionsprozess durchzuführen. Ein Extraktionsprozess fügt dem top path  $\log(\log n)$  Knoten hinzu. Er wird gestartet wenn der top path aufgrund einer *cut* Operation, weniger als  $2 \log(\log n)$  enthält, oder zum Ende der *join* Operation. Damit die Laufzeit von *cut* eingehalten werden kann, erstreckt sich ein Extraktionsprozess über mehrere *cut* Operationen, mit Fall 1. Ein Extraktionsprozess wird mit einer Folge von  $O(\log(\log n))$  Rotationen ausgeführt. Um die Invariante bezüglich der top path Länge einzuhalten, darf ein Extraktionsprozess auf so viele *cut* Operationen verteilt werden, bis mehr als  $\log(\log n)$  Knoten vom top path entfernt wurden. Der Hybrid Baum macht, mit einer Konstante, die Anzahl der durchgeführten Rotationen nach einem *cut* 1. Fall, von der Anzahl der abgespaltenen Knoten abhängig. So dass der Extraktionsprozess spätestens dann abgeschlossen wird, wenn  $\log(\log n)$  Knoten durch *cut* entfernt wurden. Kommt es zu einer



**Abbildung 3:** Mögliche Ausgangssituation vor *concatenate*.

*cut* Operationen, mit Fall 2 wird ein aktiver Extraktionsprozess abgebrochen. Von den Hilfsbäumen wird eine *split* Operation wie vom Hilfsbaum des Tango Baumes verlangt. Diese darf jedoch nur Rotationen verwenden. Für den Rot Schwarz Baum ist eine solche Variante in [1] dargestellt. Sei  $H$  eine Repräsentation des preferred path  $P_p = v_1^*, v_2^*, \dots, v_m^*$  und  $i \in \{1, 2, \dots, m\}$ . Sei  $n_1 = \log_2(\log_2(n))$ . Sei  $v_i$  der Elternknoten der Wurzel des bottom tree, mit  $d = v_i.depth$ . Es müssen der Reihe nach die Knoten mit *depth* Wert  $d + 1, d + 2, \dots, d + n_1$  extrahiert werden, damit ein valider top path entsteht.  $P_p$  wird in **zig Segmente** und **zag Segmente** unterteilt. zig Segmente entsprechen den längst möglichen Pfaden von Knoten mit rechten Kindern in  $P_p$ . zag Segmente entsprechen den längst möglichen Pfaden von Knoten mit linken Kindern in  $P_p$ . Das Blatt in  $P_p$  wird dem Segment seines Vaters zugeordnet. In Abbildung 8 sind zig und zag Segmente dargestellt. Sei  $S_{zig}$  die Folge der in zig Segmenten enthaltenen Knoten, aufsteigend sortiert nach der Tiefe und  $S_{zag}$  die Folge der in zag Segmenten enthaltenen Knoten, 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 8. 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 Schlüsselmenge, von Pfaden in  $P_p$  mit Endknoten  $p_m^*$  abgeleitet werden, siehe Abbildung 8, wobei  $l$  in  $S_{zig}$  enthalten sein muss und  $r$  in  $S_{zag}$ . Sei  $[l, r]$  ein so erstelltes Intervall zu dem Pfad von  $v_{i+n_1+1}^*$  zu  $p_m^*$ .





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

Es müssen genau die Knoten im bottom tree zum extrahieren vorbereitet werden, deren Schlüssel nicht in  $[l, r]$  liegt. Der Knoten mit Schlüssel  $l$  sowie dessen Vorgänger  $v_l$  sowie der Knoten mit Schlüssel  $r$  und dessen Nachfolger  $v_r$  werden gefunden wie in Abschnitt ?? gezeigt. Sei  $l'$  der Schlüssel von  $v_l$  und  $r'$  der Schlüssel von  $v_r$ . Mit zwei *split* 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 zwei in Abbildung 5 ist damit bereits vollzogen. In Schritt drei werden  $B$  und  $E$  nun zu BST in Listendarstellung gewandelt. Sei  $B'$  bzw.  $E'$  der Baum der dadurch von  $B$  bzw.  $E$  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 Linksrotationen 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 dann wird eine Rechtsrotation auf das linke Kind der Wurzel ausgeführt. Schritt zwei wird so oft wiederholt bis  $B'$  erzeugt ist. Abbildung 6 stellt diesen Vorgang dar. Da  $l$  der Schlüssel eines Knoten in  $S_{zig}$  ist, müssen alle Schlüssel aus  $B'$  auch als Schlüssel von Knoten in  $S_{zig}$  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, ist der Vorgang beendet.

Muss ein Extraktionsprozess aufgrund *cut* Fall 2 vorzeitig abgebrochen werden, werden aus  $B'$  und  $E'$  wieder balancierte BST erzeugt, simultan zu der Beschreibung in *join*. Im Anschluss wird mit zwei *tangoJoin* 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 der zusätzliche Zeiger in  $w$ , auf den Knoten, auf dem die nächste Rotation ausgeführt 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.

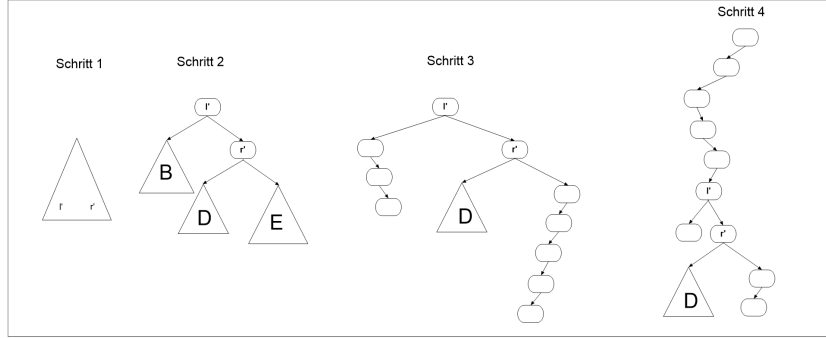
Bei Schritt eins und zwei entstehen Kosten von  $O(\log(\log n))$ , vergleichbar mit Kapitel ?? . Beim erstellen von  $B'$  aus  $B$  ist jeder Knoten an maximal drei Rotationen beteiligt, vergleiche Abbildung 6. Da maximal  $\log(\log n)$  Knoten vorbereitet werden, entstehen Kosten von  $O(\log(\log n))$ . 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  $O(k)$ .

### 1.1.3 Laufzeitanalyse beim Hybrid Baum

Zunächst wird ein Lemma benötigt.

**Lemma 1.1.** *Sei  $T$  ein Hybrid Baum mit  $n$  Knoten und  $P$  ein dazu erstellter Referenzbaum. Sei  $v$  ein Knoten aus  $T$ , dann gilt  $depth(v) = O(depth(v^*))$ .*

*Beweis.* Seien  $P_1, P_2, \dots, P_m$  die preferred path, die Knoten aus dem Pfad  $P_{v^*}$  von der Wurzel von  $P$  zu  $v^*$  enthalten. Jeder von ihnen wird nun als BST betrachtet. Für  $i \in \{1, 2, \dots, m\}$  sei  $H_i$  die Pfadrepräsentation zu  $P_i$ . Sei  $p_i^*$  der Knoten in  $P_i$  mit der größten Tiefe unter den in  $P_{v^*}$  enthaltenen. Sei  $H_i$  der Hilfsbaum der  $P_i$  repräsentiert. Die folgenden Tiefenangaben werden



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

auf  $P_i$  bzw.  $H_i$  bezogen. Ist  $p_i$  im top path enthalten, haben  $p_i$  und  $p_i^*$  die gleiche Tiefe. Liegt  $p_i$  im bottom tree so muss  $\text{depth}(p_i^*) > \log(\log n)$  gelten, aufgrund der Konstruktion von  $H_i$ . Für die Tiefe jedes Knoten in  $H_i$  gilt  $O(\log(\log n))$

□

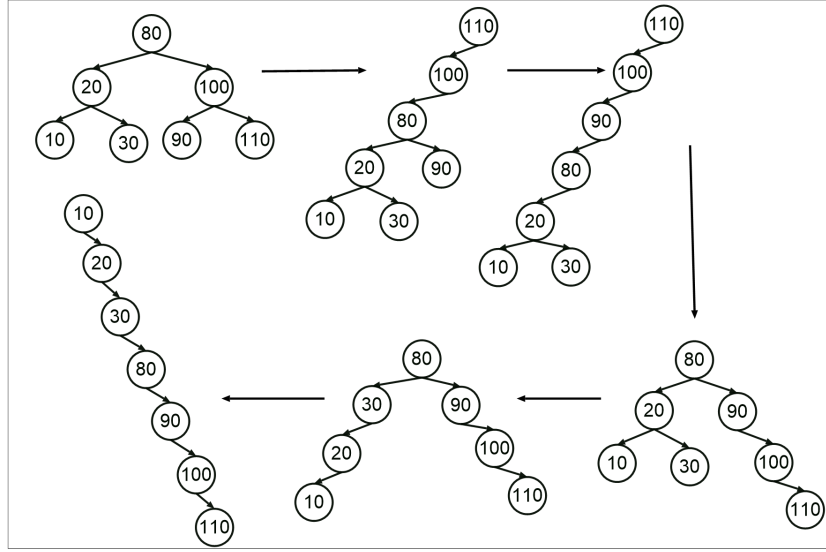
Daraus ergibt sich sofort für die Höhe eines Hybrid Baum mit  $n$  Knoten sofort  $O(\log n)$ .

In *access* entstehen, pro preferred path mit  $k$  involvierten Knoten, Kosten von  $O(k)$  wenn  $k < \log_2(\log_2(n))$  gilt und Kosten von  $O(\log_2(\log_2(n)))$  wenn  $k \geq \log(\log(n))$  gilt. Mit dem Lemma gilt deshalb für die Kosten von *access*  $O(\log(n))$ . Da pro *cut* Operation maximal Kosten von  $O(\log(\log(n)))$  entstehen, muss mit den Ausführungen zu *join*, der Hybrid Baum genau wie der Tango Baum  $\log(\log(n))$ -competitive sein.

## 1.2 Repräsentation eines preferred path beim Zipper Baum

Ziel der Repräsentation eines preferred path  $P_p = p_1^*, p_2^*, \dots, p_m^*$  ist es ohne den zusätzlichen Zeiger in den Knoten auszukommen, ohne Einschränkung bei asymptotischen Laufzeiten. Dies wird dadurch erreicht, dass der Knoten auf den in einem Extraktionsprozess die nächste Rotation ausgeführt werden muss, immer in konstanter Zeit von der Wurzel seines Hilfsbaumes  $H$  aus zugegriffen werden kann.

$H$  enthält auch einen bottom tree. Die Knoten im top path werden jedoch zu einem **zipper** angeordnet. Dieser besteht aus zwei Teilen, dem **top zipper**  $z_t$  und dem **bottom zipper**  $z_b$ .  $z_t$  und  $z_b$  dürfen jeweils maximal  $\log_2(\log_2(n))$  Knoten enthalten, gemeinsam müssen sie zumindest  $\log_2(\log_2(n))/2$  Knoten



**Abbildung 6:** Beispielhaftes überführen von  $B$  zu  $B'$ .

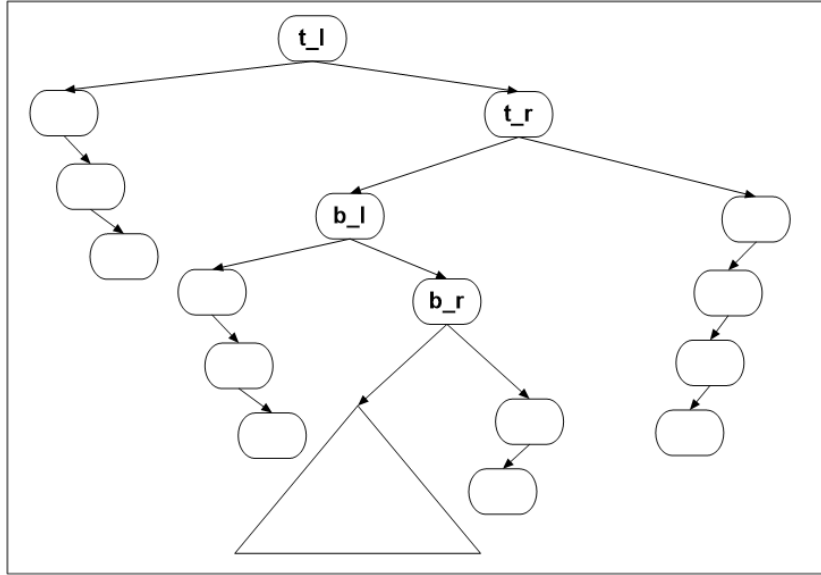
enthalten. Es wird wieder angenommen, dass ein bottom tree existiert. Sei  $n_t$  bzw.  $n_b$  die Anzahl der Knoten von  $z_t$  bzw.  $z_b$ .  $z_t$  repräsentiert Knoten  $P_t = p_1^*, p_2^*, \dots, p_{n_t}^*$  und  $z_b$  die Knoten  $P_b = p_{n_t+1}^*, p_{n_t+2}^*, \dots, p_{n_t+n_b}^*$ .  $z_t$  ist wie folgt aufgebaut:

Sei  $S_{zig}$  und  $S_{zag}$  zu  $P_t$  definiert wie in Abschnitt 1.1.2. Sei  $t_l$  bzw.  $t_r$  der Knoten in  $S_{zig}$  bzw.  $S_{zag}$  mit der kleinsten Tiefe.  $t_l$  ist die Wurzel von  $H$ .  $t_r$  ist das rechte Kind von  $t_l$ . Der linke Teilbaum von  $t_l$  hat Listenform und wird von den restlichen Knoten in  $S_{zig}$  gebildet, so dass kein Knoten in diesem Teilbaum ein linkes Kind hat. Der rechte Teilbaum von  $t_r$  hat Listenform und wird von den restlichen Knoten in  $S_{zag}$  gebildet, so dass kein Knoten in diesem Teilbaum ein rechtes Kind hat.

$z_b$  wird simultan aus  $P_b$  erzeugt und seien  $b_l$  und  $b_r$  die Knoten in  $z_b$  entsprechend zu  $t_l$  und  $t_r$  in  $z_t$ .  $b_l$  ist das linke Kind von  $t_r$ . Die Wurzel des bottom Tree ist das linke Kind von  $t_r$ . Das die Links-Rechts-Beziehung eingehalten wird ergibt aus den Aufbau der zig und zag Segmente und die Wurzel des bottom tree ist in konstanter Zeit erreichbar. Abbildung ?? zeigt eine beispielhafte Darstellung.

### 1.3 Die *access* Operation beim Zipper Baum

Der Ablauf unterscheidet sich nur in den Hilfsoperationen von dem des Hybrid Baum. Auf diese wird jeweils eingegangen. Sei  $k$  der Parameter der Operation und  $p$  der Zeiger der Operation in den BST.



**Abbildung 7:** Pfadrepräsentation beim Zipper Baum, basiert auf einer Abbildung aus [1].

**cut Operation** Auch hier gibt es die zwei Fälle von *cut* beim Hybrid Baum. Es werden zunächst die Knoten aus dem top zipper zu einem top path gewandelt, vergleichbar Abbildung 5 Schritt 4, bis der gesuchte Knoten im top path enthalten ist, oder die Suche in einem anderen Hilfsbaum weitergeführt wird. Ist der top zipper vollständig in einen top path umgewandelt, wird der Vorgang beim bottom zipper fortgesetzt. Dieser hat dann die Stellung des top zipper. Außerdem wird dann ein Extraktionsprozess angestoßen, der  $\log(\log(n))$  Knoten aus dem bottom tree auslagert, um einen neuen bottom zipper zu erzeugen. Die Konstante zur Anzahl der durchzuführenden Rotationen im Extraktionsprozess, muss so gewählt sein, dass der Prozess nach  $\log_2(\log_2(n))/2$  entfernten Knoten abgeschlossen ist. Wird auch der zweite Zipper vollständig aufgebraucht, handelt es sich um einen Fall 2 cut, ansonsten um einen Fall 1 cut.

Fall 1:

Es muss zur oberen Beschreibung nichts weiteres durchgeführt werden. Ist in  $H$  bereits zu Beginn der cut Operation ein Extraktionsprozess aktiv, kann die Wurzel  $w$  des bottom tree auch ohne zusätzlichen Zeiger in konstanter Zeit erreicht werden. Zum einhalten der Laufzeit, muss jedoch jede Rotation auf Knoten ausgeführt werden, die in konstanter Zeit von  $w$  aus erreichbar sind. Zusätzlich zur Beschreibung aus 5 benötigt man hierzu noch eine entsprechende *split* Operation. Eine in [1] vorgestellte *split* Operation, erfüllt diese Eigenschaft.

Fall 2:

Auch hier werden die Knoten der Zipper als balancierte Bäume mit einem Knoten betrachtet und vorgegangen wie beim Hybrid Baum.

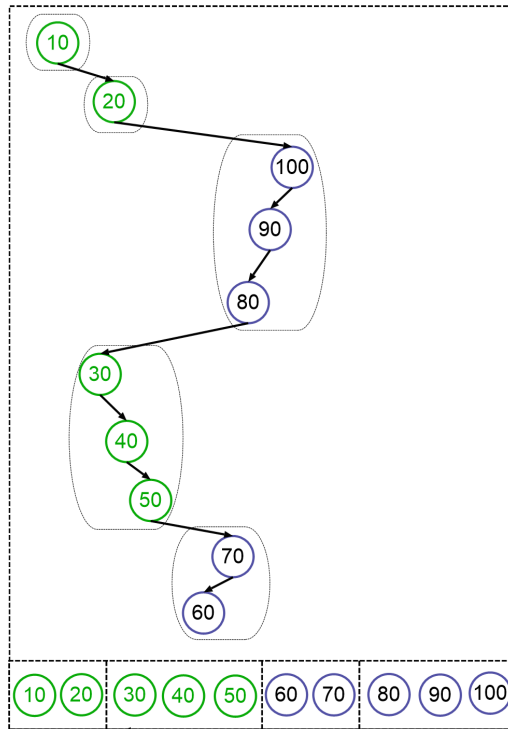
Aus der Beschreibung folgt, dass diese Operation die gleich asymptotische Laufzeit, wie die *cut* Operation des Hybrid Baum besitzt.

**join Operation** Diese Operation verhält sich zunächst wie *join* beim Hybrid Baum, erstellt dann jedoch anstatt eines top path den zipper. An Abbildung 5 und der Beschreibung dazu ist direkt erkennbar, dass dies keinen Einfluss auf die asymptotische Laufzeit hat.

**extract Operation** Hier entfällt im Vergleich zum Hybrid Baum der letzte Schritt aus Abbildung 5.

## 1.4 Laufzeitanalyse beim Zipper Baum

Aus den oberen Ausführungen folgt, dass der Zipper Baum genau wie der Hybrid Baum  $\log(\log(n))$ -competitive ist und die worst case Laufzeit von *access*  $O(\log(n))$  ist.



**Abbildung 8:** 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.