Bachelorarbeit

Andreas Windorfer

31. Juli 2020

Inhaltsverzeichnis

1 Splay Baum

Der in [1] vorgestellte Splay Baum ist ein online BST der ohne zusätzliche Hilfsdaten in seinen Knoten auskommt. Nach einer $access(k)$ Operation, ist der Knoten mit Schlüssel k die Wurzel, des Splaybaum. Es gibt keine Invariante, welche eine bestimmte maximale Höhe garantiert. Splay Bäume können sogar zu Listen entarten. Amortisiert betrachtet verfügen sie dennoch über sehr gute Laufzeiteigenschaften.

1.1 *access* Operation

Die wesentliche Arbeit leistet eine Hilfsoperation namens *splay*. Nach deren Ausführung befindet sich der Knoten mit dem gesuchten Schlüssel k an der Wurzel und es wird nur noch eine Referenz auf ihn zurückgegeben.

***splay* Operation** Sie p der Zeiger der Operation in den BST. Zunächst wird eine gewöhnliche Suche ausgeführt bis p auf den Knoten v mit Schlüssel k zeigt. Nun werden iterativ sechs Fälle unterschieden bis v die Wurzel des Baumes darstellt. Zu jedem Fall gibt es einen der Links-Rechts-Symmetrisch ist. Sei u der Elternknoten von v.

1. v ist das linke Kind der Wurzel (zig-Fall):
   Es wird eine Rechtsrotation auf v ausgeführt. Nach dieser ist v die Wurzel des Splaybaum und die Operation wird beendet.

2. v ist das rechte Kind der Wurzel (zag-Fall):
   Symmetrisch zu zig.

3. v ist ein linkes Kind und u ist ein linkes Kind. (zig-zig-Fall):
   Dieser Fall unterscheidet den Splaybaum vom einem anderen BST (move-to-root), mit schlechteren Laufzeiteigenschaften. Es wird zuerst eine Rechtsrotation auf u ausgeführt und erst danach eine Rechtsrotation auf v. Bei move-to-root ist es genau anders herum.

4. v ist ein rechtes Kind und u ist ein rechtes Kind. (zag-zag-Fall):
   Symmetrisch zu zig-zig.

5. v ist ein linkes Kind und u ist ein rechtes Kind. (zig-zag-Fall):
   Es wird eine Rechtsrotation auf v ausgeführt. Im Anschluss wird eine Linksrotation auf u ausgeführt.

6. v ist ein rechtes Kind und u ist ein linkes Kind. (zag-zig-Fall):
   Symmetrisch zu zig-zag.

Abbildung 1 zeigt drei der Fälle. Trotz der Einfachheit kann die Auswirkung einer einzelnen *splay* Operation sehr groß sein. Abbildung 2 aus [1] zeigt eine solche Konstellation.
Die Laufzeit von *access* auf einem BST mit n Knoten ist $O(n)$.

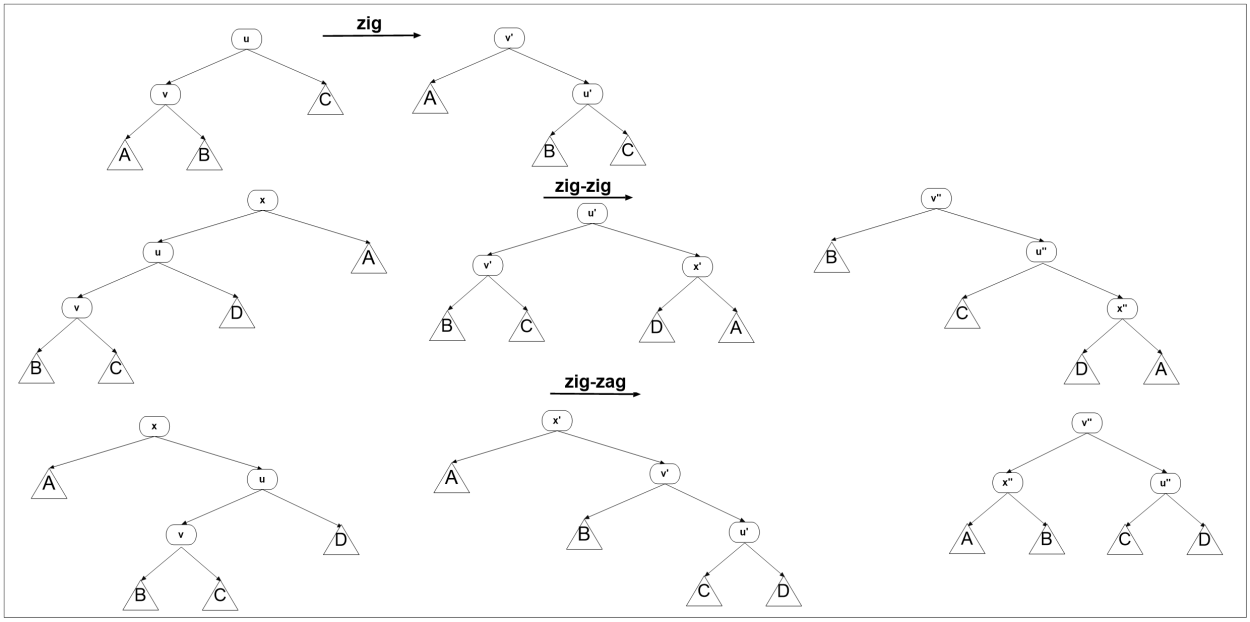

Abbildung 1: Darstellung von zig, zig-zag und zig-zig.

1.2 Amortisierte Laufzeitanalyse von *splay*

Es wird die Potentialfunktionsmethode aus Kapitel **??** verwendet. Sei v ein Knoten im Splay Baum T. Eine Funktion $w(v)$ liefert zu jedem Knoten eine reelle Zahl > 0, die Gewicht genannt wird. Eine Funktion $tw(v)$ bestimmt die Summe der Gewichte aller im Teilbaum mit Wurzel v enthaltenen Knoten. Der Rang $r(v)$ ist definiert durch $r(v) = \log_2(tw(v))$. Sei V die Menge der Knoten von T. Als Potentialfunktion wird

$$\Phi = \sum_{v \in V} r(v)$$

verwendet.

Access Lemma 1.1. *Sei T ein Splay Baum mit n Knoten und Wurzel w und einem Knoten v mit Schlüssel k. Es werden den Knoten fest zugeordnete Gewichte angenommen. Die amortisierte Laufzeit von splay(k) ist maximal* $3(r(w) - r(v)) + 1 = O(\log(tw(w))/tw(v)) = O(\log(n))$

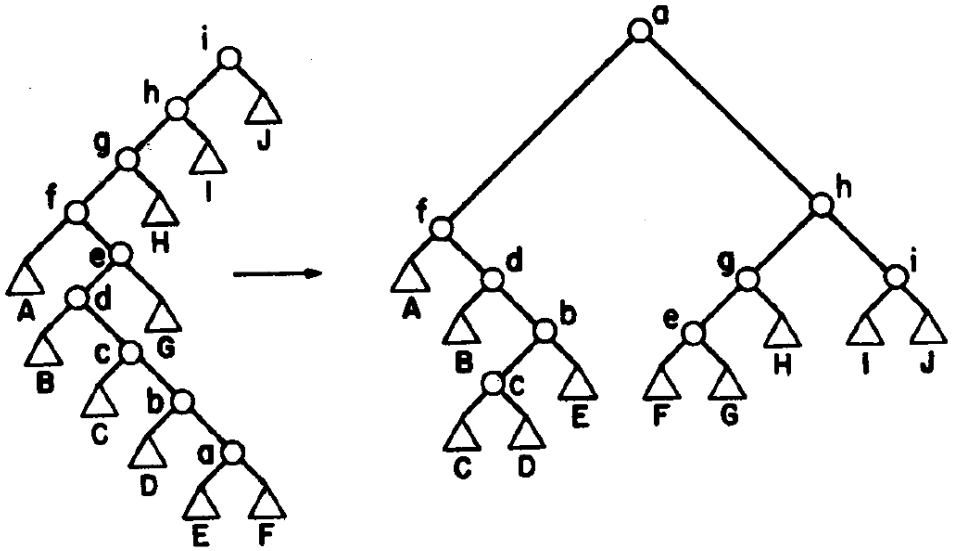

FIG. 4. Splaying at node *a*.

Abbildung 2: Eine einzige *splay* Operation. [1]

Beweis. Zunächst wird für *zig*, *zig-zag* und *zig-zig* gezeigt, dass die amortisierten Kosten nicht größer als $3\left(r\left(v\right)' - r\left(v\right)\right) + 1$ sind. Für die anderen drei Fälle folgt es dann aus der Symmetrie. Im Anschluss wird die gesamte Operation betrachtet. An Abbildung 1 ist zu erkennen, dass sich der Wert von $tw\left(\right)$ nur an den Knoten v, dessen Elternknoten u und dem Elternknoten x von u verändern kann. Damit gilt

$$\Phi' - \Phi = r\left(u\right)' + r\left(v\right)' + r\left(x\right)' - r\left(u\right) - r\left(v\right) - r\left(x\right)$$

zig In diesem Fall existiert x nicht, damit gilt
$\Phi' - \Phi = r\left(u\right)' + r\left(v\right)' - r\left(u\right) - r\left(v\right)$. Der Wert von $tw\left(\right)$ für die Wurzel ist unabhängig von Zustand des Splay Baum, da an ihr alle im Baum vorhandenen Gewichte aufsummiert werden. Deshalb muss $tw\left(v\right)' = tw\left(u\right)$ gelten. Daraus folgt $\Phi' - \Phi = r\left(u\right)' - r\left(v\right)$. Aus $r\left(v\right)' \geq r\left(u\right)'$ und $r\left(v\right)' \geq r\left(v\right)'$ folgt $\Phi' - \Phi \leq r\left(v\right)' - r\left(v\right) \leq 3\left(r\left(v\right)' - r\left(v\right)\right)$. Addieren von 1 aufgrund der Rotation ergibt Kosten $\leq 3\left(r\left(v\right)' - r\left(v\right)\right) + 1$.

zig-zig Es müssen zwei Rotationen ausgeführt werden, deshalb entstehen amortisierte Kosten von

$$\begin{aligned}
&2 + r(u)' + r(v)' + r(x)' - r(u) - r(v) - r(x) && \text{mit } r(x) = r(v)' \\
=&2 + r(u)' + r(x)' - r(u) - r(v) && \text{mit } r(v)' \geq r(u)' \text{ und } r(u) \geq r(v) \\
\leq&2 + r(v)' + r(x)' - 2r(v)
\end{aligned}$$

Nun wird zunächst die Behauptung aufgestellt, dass dieser Ausdruck klein genug ist, dies wird dann über Äquivalenzen gezeigt.

$$\begin{aligned}
&2 + r(v)' + r(x)' - 2r(v) \leq 3\left(r(v)' - r(v)\right) \\
\Leftrightarrow&2 \leq 2r(v)' - r(x)' - r(v) \\
\Leftrightarrow&-2 \geq -2r(v)' + r(x)' + r(v) \\
\Leftrightarrow&-2 \geq \log_2(tw(x')/tw(v')) + \log_2(tw(v)/tw(v'))
\end{aligned}$$

Dass die letzte Ungleichung gilt, kann man an einer Eigenschaft des $\log_2$ ableiten. Für $a, b \in R$ mit $a, b > 0$ und $a + b \leq 1$ gilt $\log_2(a) + \log_2(a) \leq -2$. An Abbildung 1 ist zu erkennen, dass sich $tw(v)$ vom Ausgangszustand zum Zwischenzustand hin nicht verändert. $tw(x')$ ist ebenfalls unverändert zum Zwischenschritt. Es kann also bei beiden Knoten mit den Werten aus dem Zwischenschritt gearbeitet werden. $tw(v') = tw(x') + tw(v) + w(u)$. Daraus folgt $((x') + (v))/tw(v') < 1$ und mit der Eigenschaft von $\log_2$ folgen die Ungleichungen.

zig-zag

$$\begin{aligned}
&2 + r(u)' + r(v)' + r(x)' - r(u) - r(v) - r(x) && \text{mit } r(x) = r(v)' \text{ und } r(v) \leq r(u) \\
\leq&2 + r(u)' + r(x)' - 2r(v)
\end{aligned}$$

Nun wird wie bei zig-zig vorgegangen.

$$\begin{aligned}
&2 + r(u)' + r(x)' - 2r(v) \leq 2\left(r(v)' - r(v)\right) \\
\Leftrightarrow&2 \leq 2r(v)' - r(x)' - r(u)' \\
\Leftrightarrow&-2 \leq -2(v)' + r(x)' + r(u)' \\
\Leftrightarrow&-2 \geq \log_2(tw(x')/tw(v')) + \log_2(tw(u')/tw(v'))
\end{aligned}$$

Mit der $\log_2$ Regal aus zig-zig und Abbildung 1 folgt die Behauptung. Betrachtet man die Kosten der Gesamtoperation so bildet sich eine Teleskopsumme, die möglicherweise, wenn ein zig bzw. zag Fall enthalten ist, mit eins addiert werden muss. Daraus folgt das Lemma. □

1.3 Dynamische Optimalitäts Vermutung

Der Splay Baum erfüllt working set und dynamig finger aus Kapitel ?? und auch noch einige weitere in dieser Arbeit nicht aufgeführte Eigenschaften. Für dynamic finger ist ein sehr aufwändiger Beweis in [?] enthalten. working set wurde bereits in [1] gezeigt. Dieser Beweis wird hier vorgestellt. Die anderen Eigenschaften (außer dynamisch optimal) aus Kapitel ?? folgen dann aus diesen beiden. Aufgrund dieser oberen Schranken wurde in [1] die Vermutung aufgestellt, dass der Splay Baum dynamisch optimal ist. Bewiesen ist bisher nur, dass er $\log(n)$-competitive ist, dies folgt aus dem access lemma. Würde für eine solche obere Schranke gezeigt werden, dass der Splay Baum diese nicht einhalten kann, jedoch ein anderer BST schon, wäre die Vermutung zur dynamischen Optimalität widerlegt. Auch das ist bis heute nicht geschehen.

working set 1.1. *Es sei T ein Splaybaum mit n Knoten. Sei $X = x_1, x_2, .., x_m$ eine für T erstellte Zugriffsfolge, mit $m \geq n$ und $m \geq n \log_2(n)$. Sei $w_i = |\{x_j | t_{xi} < j \leq i\}|$ definiert, wie in Kapitel ??. Dann gilt für die amortisierte Laufzeit $O(\sum_{i=1}^{m} \log w_i)$.*

Beweis. Den Knoten werden die Gewichte $1, 1/2^2, 1/2^3.., 1/2^n$ zugeordnet. Diesmal ist die Zuordnung nicht fest. Nach jeder *access* Operation können sich Gewichte ändern. Sei i der kleinste Index mit $x_i = key(v)$ für einen Knoten v. Sei $a = |\{x_l | l < i\}|$, dann wird v zum Start das Gewicht $1/2^{a+1}$ zugeordnet. Auf Knoten auf deren Schlüssel nicht zugegriffen wird, verteilen sich die kleinsten Gewichte beliebig. $tw(v)$ und Φ sind definiert wie beim access lemma.
Nach einem Zugriff x_j werden die Gewichte neu zugeordnet. Der Knoten v_j mit Schlüssel x_j erhält das Gewicht $1/1$. Sei $1/k^2$ das Gewicht von v_j vor *access* (x_j). Sei u ein Knoten mit $w(u) = 1/k^*$ mit $k^* \in \{1, 4, .., (k-1)^2\}$ direkt vor dem Zugriff x_j. Dann ist $1/(k^*+1)^2$ das Gewicht von u nach dem Zugriff x_j. Zu beachten ist, dass nach einer solchen Neuzuordnung die Menge der im Baum enthaltenen Gewichte unverändert bleibt.
Diese Verteilung der Gewichte garantiert, dass direkt vor Zugriff x_j, v_j ein Gewicht von $1/{w_i}^2$ hat, somit gilt $tw(v_j) \geq 1/{w_i}^2$. Der Wert von tw für die Wurzel r in T direkt vor der Ausführung von x_j ist $W = \sum_{i=1}^{n} 1/n^2 < 2 =$. Diese Werte in Access Lemma eingesetzt ergibt Kosten von $O(\log({w_i}^2/W))$, mit $w_i/W = O(1)$. Durch die nachfolgende Neuzuordnung der Gewichte kann Φ nur kleinere Werte annehmen, denn nur das Gewicht der neuen Wurzel erhöht sich, tw ist für die Wurzel aber konstant.
Über die gesamte Zugriffssequenz kann das Potential nicht um mehr als $\sum_{i=1}^{n} \log_2(w(i)/W)$ kleiner werden. Denn W ist der maximale Wert von $tw()$ und der minimale ist $1/n^2$. Daraus folgt eine maximale Verringerung

des Potentials von $= (n \log(n))$. $\square$

Literatur

[1] Daniel Dominic Sleator and Robert Endre Tarjan. Self-adjusting binary search trees. *J. ACM*, 32(3):652–686, July 1985.