Bachelorarbeit

Andreas Windorfer

10. Juli 2020

Inhaltsverzeichnis

1 Splaybaum

Der in [1] vorgestellte Splaybaum ist ein online BST der ohne zusätzliche Hilfsdaten in seinen Knoten auskommt. Nach einer *access* Operation, ist der Knoten mit Schlüssel k die Wurzel, des Splaybaum. Es gibt keine Invariante, welche eine bestimmte maximale Höhe garantiert. Splaybäume 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 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 Vater 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 linke Kind der Wurzel (zag-Fall):
   Symmetrisch zu zig.

3. 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.

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

5. 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.

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

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 $O(n)$.

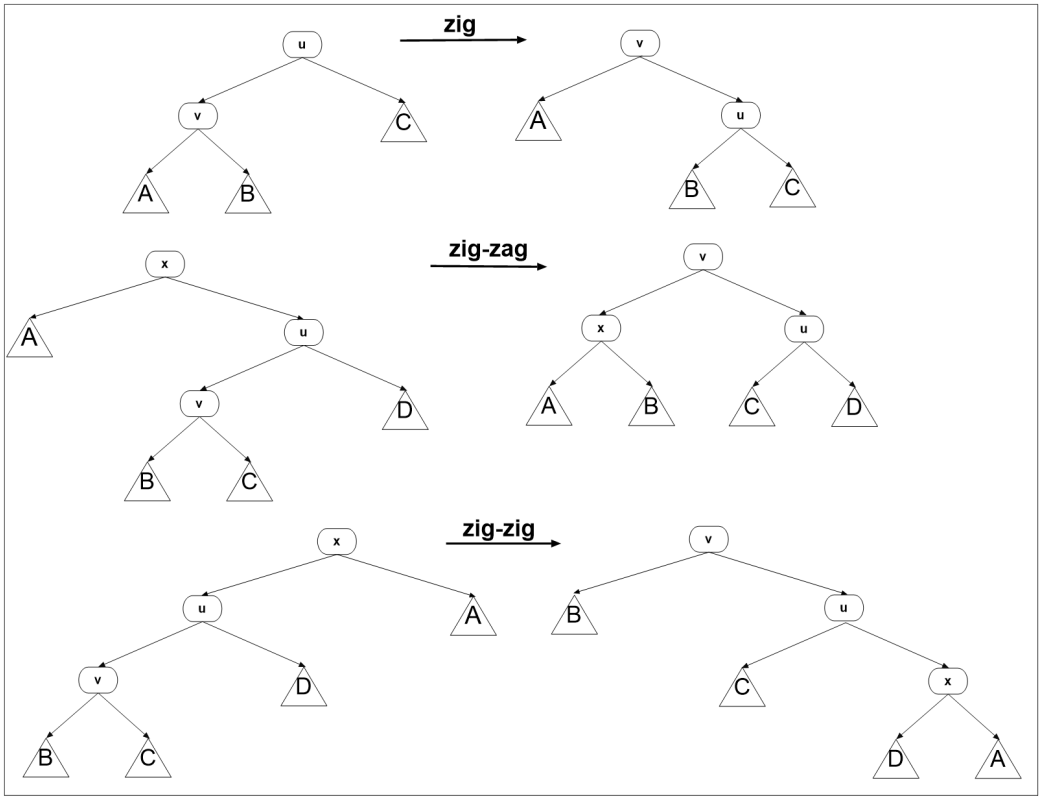

Abbildung 1: Derstellung von zig, zig-zag und zig-zig.

1.2 Amortisiert Laufzeitanalyse von *splay*

Es wird die Potentialfunktionsmethode aus Kapitel **??** verwendet. Sei v ein Knoten im Splaytree T. Eine Funktion $w(v)$ liefert zu jedem Knoten eine reelle Zahl > 0, die Gewicht genannt wird. Das Gewicht eines Knotens ist unveränderlich. Eine Funktion $tw(v)$ bestimmt die Summe aller Gewichte im Teilbaum mit Wurzel v. 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.

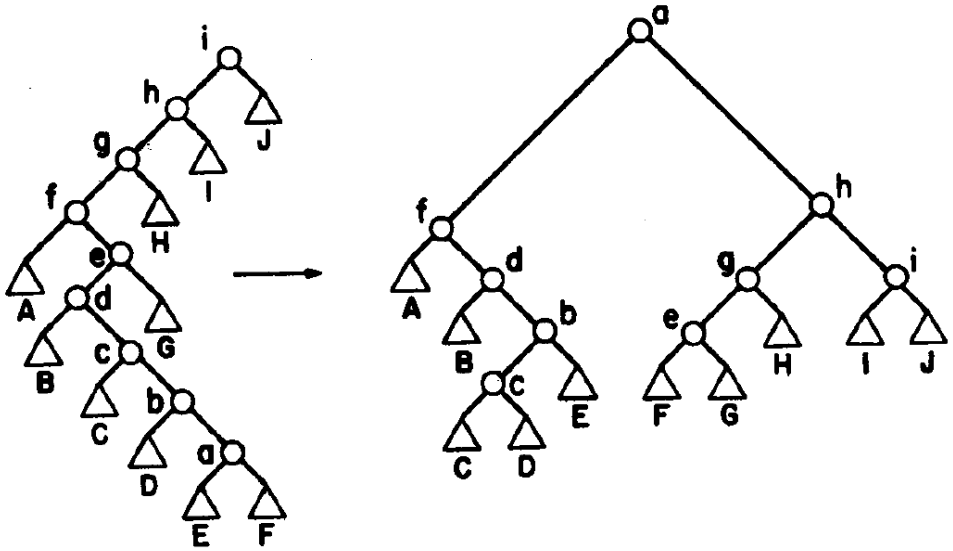

FIG. 4. Splaying at node *a*.

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

Access Lemma 1.1. *Es sei T ein BST mit Knoten u, v so, dass u ein Kind von v ist. T' ist der BST, der durch ausführen der Rotation $(key(u), key(v))$ aus T entsteht. Gilt $key(u), key(v) \in [l, r]$, dann ist T'^r_l der BST der aus T^r_l durch Ausführen von $(key(u), key(v))$ entsteht. Anderenfalls gilt $T'^r_l = T^r_l$.*

Beweis. □

Literatur

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