Bachelorarbeit

Andreas Windorfer

17. Juli 2020

Inhaltsverzeichnis

1 Multisplay Baum

Der Multisplay Baum ist eine Variation zum Tango Baum. Ein preferred path wird hier durch einen Splaybaum dargestellt. Amortisiert betrachtet, ist er $\log(\log(n))$-competitive und garantiert $O(\log(n))$ im worst case, bei einer einzelnen *access* Operation. n steht wieder für der Anzahl der Knoten von T. Da der Splaybaum kein balancierter Baum ist, gibt es zusätzliche mögliche Zustände im Vergleich zu einem Tango Baum mit der gleichen Knotenzahl.

Auch der Multisplay Baum verwendet einige Hilfsdaten je Knoten. Zum

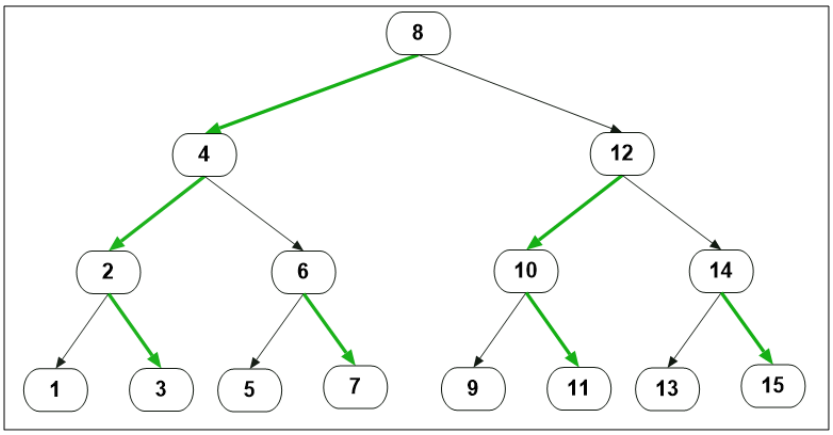

Abbildung 1: Refernzebaum mit grün gezeichneten preferred paths

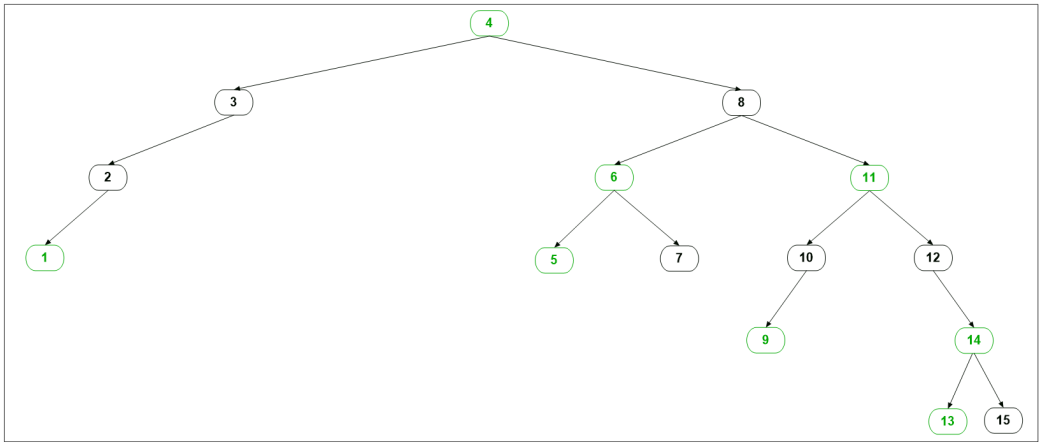

Abbildung 2: Refernzebaum mit grün gezeichneten preferred paths

einen bereits bekannte Variablen bzw. Konsanten *isRoot*, *depth* und *minDepth*. Aber auch welche die beim Tango Baum nicht verwendet sind. Sei v

ein Knoten in T und u der Knoten mit $key\,(v) = key\,(u)$ im Referenzbaum P. Sei H der Hilfsbaum der v enthält. Die Konstante *height* hat den Wert der Höhe von u. Die Variable *treeSize* enthält die Anzahl der Knoten von H.

1.1 Die *access* Operation beim Multisplay Baum

Zu beachten ist, dass jede BST Darstellung auch eine Splaybaum Darstellung ist. Anders als beim Tango oder Zipper Baum, muss ein neu erzeugter Hilfsbaum also nicht so angepasst werden, dass er weitere Invarianten einhält. Sei v ein Knoten in T, dann wird mit v^* wieder der Knoten in P bezeichnet, mit $key\,(v) = key\,(v^*)$. Nach einer $access(k)$ Operation ist der Knoten v_k mit dem Schlüssel k die Wurzel von T. Zunächst wird eine gewöhnliche Suche in T durchgeführt, bis der Zeiger p der Operation auf v_k zeigt. Eine Abweichung zu den preferred path des Tango Baum ist, dass das preferred child des Knoten mit Schlüssel k zunächst unverändert bleibt. Ist v_k gefunden, werden die Pfadrepräsentationen aktualisiert. Dabei wird bottom up vorgegangen. Damit v_k zur Wurzel von T wird, wechselt das preferred child von ${v_k}^*$ immer. Das ist ein Unterschied zum Tango Baum, bei dem dieses nur wechselt, wenn zu Beginn, das rechte Kind das preferred child war. Die aufgrund von ${v_k}^*$ notwendige Aktualisierung der Pfadrepräsentation wird als letztes durchgeführt. In den Beschreibungen von *cut* und *join* wird von einem zugrunde liegenden preferred child Wechsel vom linken Kind zum Rechten ausgegangen. Der andere Fall ist symmetrisch.
Sei $P_p = {q_1}^*, {q_2}^*, .., {q_m}^*$ ein preferred path und ${q_i}^*$, mit $im \in \{1, 2, .., m\}$, der Knoten an dem das preferred child wechselt. Sei k der Schlüssel von q_i. Sei A der Hilfsbaum der P_p repräsentiert. Sei $U = \{q_1, q_2, .., q_i\}$ und $L = \{q_{i+1}, q_{i+2}, .., q_m\}$. Sei q_r das rechte Kind von q_i und B der Hilfsbaum in dem q_r enthalten ist.

***cutJoin* Operation beim Multisplay Baum** Beim Multisplay Baum werden *cut* und *join* zu einer Operation zusammengefasst. Abbildung **??** stellt die Zusammenhänge der Schlüssel dar. Das Vorgehen ist sehr ähnlich zu dem aus **??** sehr ähnlich, weshalb hier weniger detailliert darauf eingegangen wird. Zunächst wird die *isRoot* Variable von der Wurzel von A auf *false* gesetzt und $splay(k)$ auf A ausgeführt. Dadurch entsteht ein Hilfsbaum C mit Wurzel q_i Sei C_L der linke Teilbaum von q_i und C_R der Rechte.
Es wird der Knoten $q_{l'}$, mit dem kleinsten Schlüssel $l' > key(q_i)$, aus U benötigt. $q^*_{l'}$ muss eine kleinere Tiefe als q^*_i haben. Deshalb kann $q^*_{l'}$ gefunden werden indem wie folgt vorgegangen wird. p muss auf die Wurzel von C_L gesetzt werden. In einer Schleife wird p so oft auf das linke Kind p_l von p gesetzt, bis der Wert der *minDepth* Variable von p_l größer als die Tiefe von

q_i^* ist.
Nun wird $splay(l')$ auf C_l ausgeführt. Nach dieser Operation muss bei der Wurzel des rechten Teilbaumes von $q_{l'}$ *isRoot* noch auf *true* gesetzt werden. Abbildung 3 stellt es nochmals dar. Existiert $q_{l'}$ nicht, entfällt die zweite *splay* Operation und es wird *isRoot* der Wurzel von C_L auf *true* gesetzt. Der *cut*-Teil ist nun abgeschlossen und wir kommen zum *join*-Teil.
Es wird der Knoten $q_{r'}$, mit dem kleinsten Schlüssel $r' > key(q_i)$, aus U benötigt. $q_{r'}^*$ muss eine kleinere Tiefe als q_i^* haben. Deshalb kann $q_{r'}^*$ gefunden werden indem wie folgt vorgegangen wird. p muss auf die Wurzel von C_R gesetzt werden. In einer Schleife wird p so oft auf das rechte Kind p_r von p gesetzt, bis der Wert der *minDepth* Variable von p_r größer als die Tiefe von q_i^* ist.
Es wird $splay(r')$ auf C_R ausgeführt. Nun wird der linke Teilbaum D_L von q_r betrachtet. Jeder Knoten aus R muss in D_L enthalten sein, denn für $v_R \in R$ gilt $k < key(v_R) < r'$. Kein Knoten aus $L \cup U$ kann in D_L enthalten sein, da für $v_L \in L$, $key(v_L) < k$ und für für $v_U \in U$, $key(v_U) > r'$ gilt. Somit muss das linke Kind von $q_{r'}$ die Wurzel von B sein. Die *isRoot* Variable dieses Knotens wird auf *false* gesetzt.

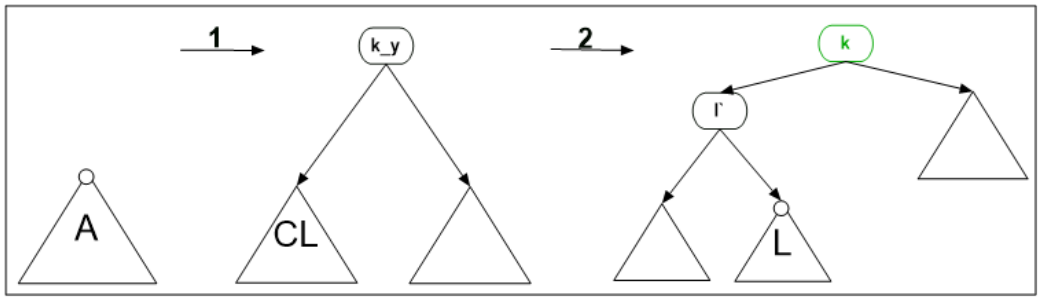

Abbildung 3: Ablauf zum erzeugen einer neuen Pfadrepräsentation, nach einem preferred child Wechsel vom linken Kind zum Rechten.

Literatur