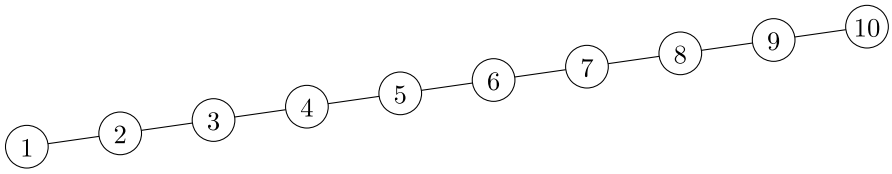
RWTH Aachen **SS 2016**
Lehrgebiet Theoretische Informatik Lösungsvorschlag 4
Rossmanith–Hensel–Kuinke–Sánchez 9. Mai 2016

Übung zur Vorlesung Datenstrukturen und Algorithmen

Aufgabe T11

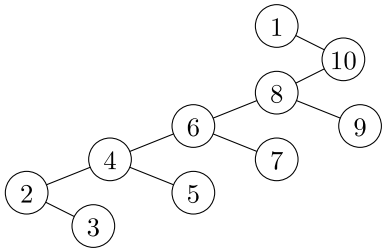
a) Bei jeder Einfügeoperation wird der eingefüge Schlüssel einmal nach links rotiert (einfaches *zag*). Nach dem Einfügen der Schlüssel $1, \ldots, 10$ sieht der Baum so aus:



Da jedes Einfügen in konstanter Zeit geschieht, ist die Laufzeit für das Einfügen der Schlüssel $1, \ldots, n$ linear.

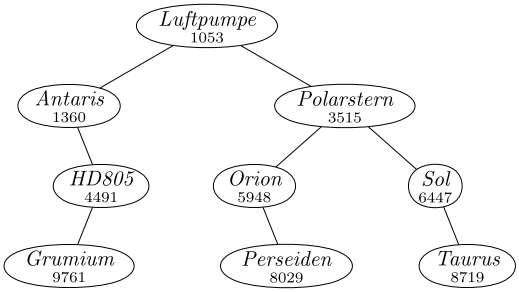
b) Wird ein Schlüssel, der bereits in der Wurzel steht, ein zweites Mal eingefügt, wird die Wurzel ersetzt und es muß nicht rotiert werden. Deshalb ist das Ergebnis das gleiche.

c)

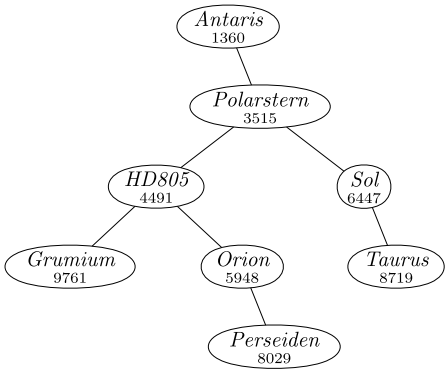


Aufgabe T12

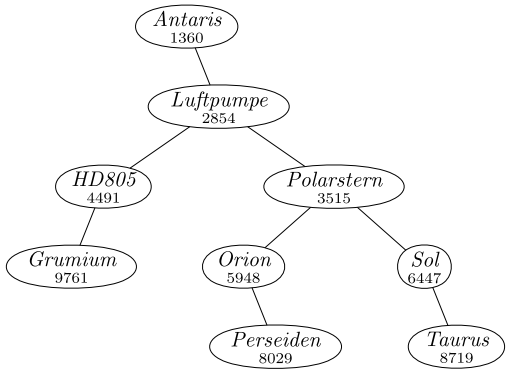
a)



b)

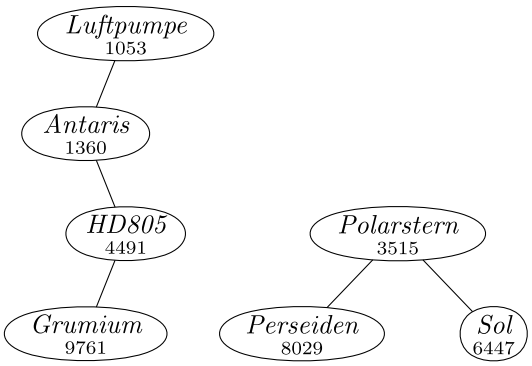


c)



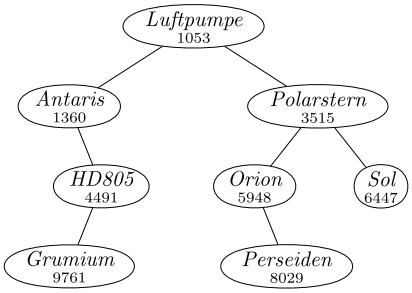
Aufgabe T13

Ist der Schlüssel s vorhanden, wird er zunächst gelöscht. Danach wird s mit der Priorität $-\infty$ (bzw. einer Priorität, die niedriger ist als die aller anderen Knoten) neu eingefügt und dabei natürlich zur Wurzel durchrotiert. Nun ergeben der linke und rechte Unterbaum die gewünschten Treaps. Die Laufzeit für das Löschen und Einfügen von s ist linear in der Höhe des ursprünglichen Treaps.

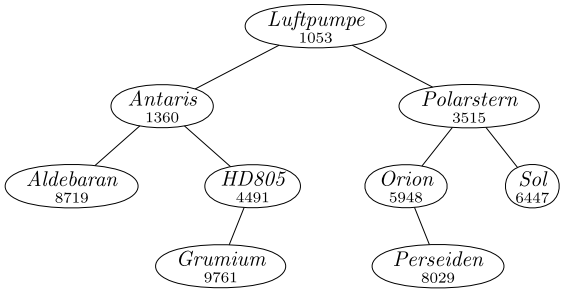


Aufgabe H11

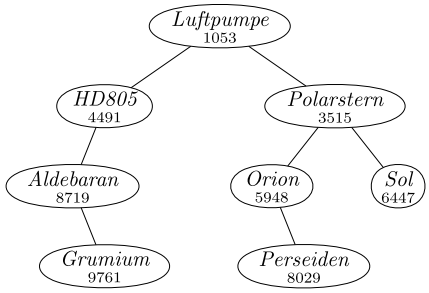
Der Treap sah so aus:



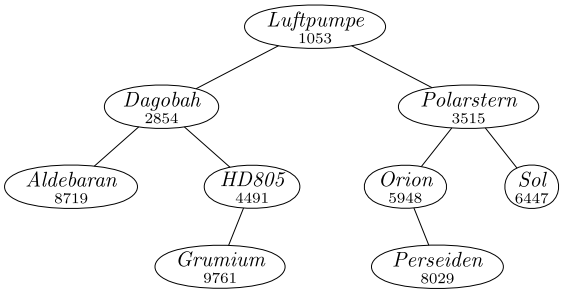
a)



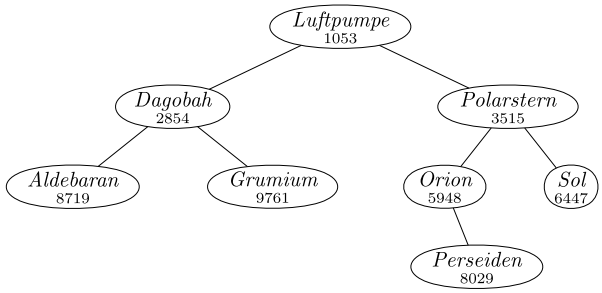
b)



c)



d)



Aufgabe H12

Wenn nach 588 gesucht wird: zag-zig, zig-zig, zag-zag, zag-zig

Wenn nach 730 gesucht wird: zig-zig, zag-zig, zag-zig, zig-zag

Wenn die 650 gelöscht wird: zag-zag, zig

Aufgabe H13

Wir können einfach auf B ein $splay(k)$ ausführen, was den Schlüssel k in die Wurzel befördert, oder, falls k in B nicht vorkommt den nächstkleineren oder nächstgrößeren Schlüssel.

Jetzt können wir einfach aus dem linken Unterbaum der Wurzel B_1 machen und aus dem rechten Unterbaum B_2.

Was passiert wenn wir folgende Operationen in beliebiger Reihenfolge und Kombination durchführen: Einfügen, Löschen, Suchen und *split*?

Da wir es mit mehreren Splaybäumen zu tun haben, können wir als Potentialfunktion einfach die Summe der Potentialfunktionen aller Bäume nehmen. Dann sieht man leicht, daß die amortisierten Kosten $O(\log n)$ sind, weil bei einem *split* nur eine *splay*-Operation ausgeführt wird und anschließend beim Zerteilen in zwei Bäume die Gesamtpotentialfunktion sogar abnimmt.