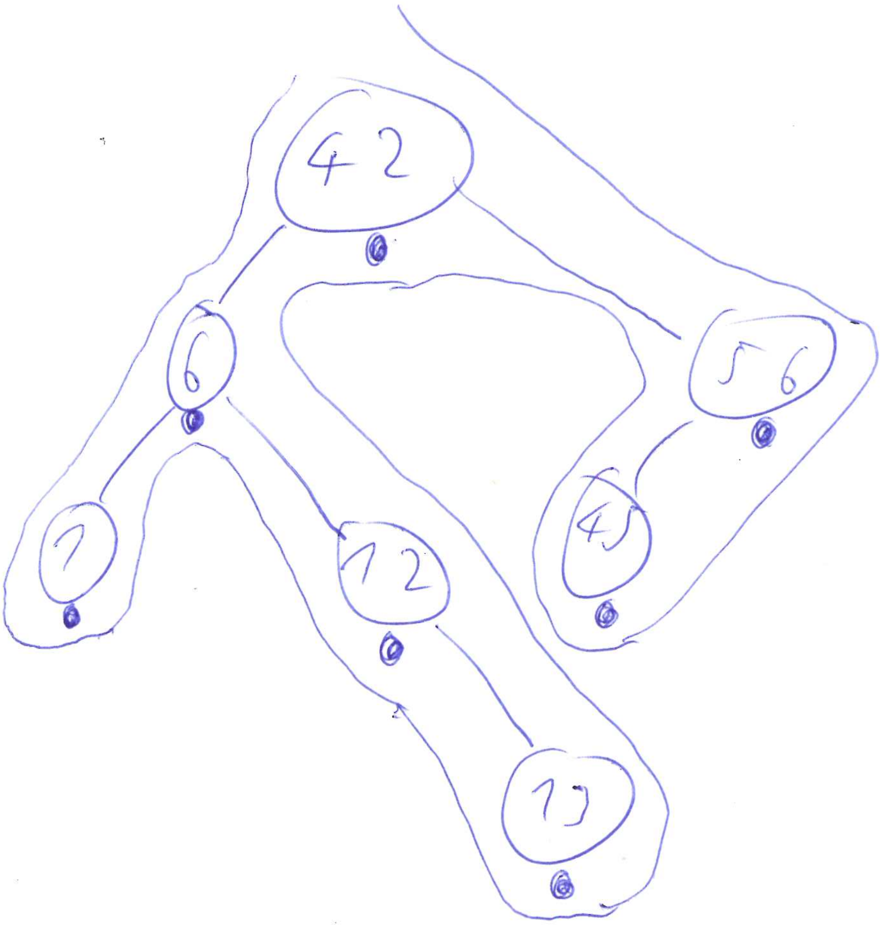
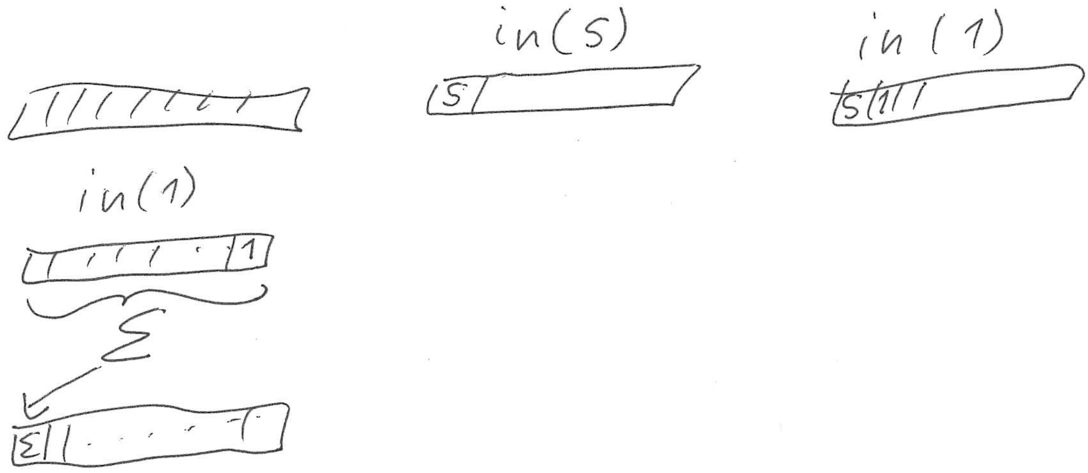


1 6 12 13 42 45 56

Array Größe m.

In jedem Schritt wird ein neues Element eingefügt. Wenn voll, summiere auf, schreibe Summe in erstes Element, lösche den Rest.



$\Phi(n)$: Anzahl ~~freier~~ belegter Plätze zum Zeitpunkt n
multipliziert mit Konstante c

$t(n)$: Laufzeit ~~dem~~ für Schritt n

$t(n) + \Phi(n) - \Phi(n-1)$

1. Fall: Array nicht voll

$t(n) = 1$

$\Phi(n) - \Phi(n-1) = 1$

$t(n) + \Phi(n) - \Phi(n-1) = O(1)$

2. Fall: Array voll

$t(n) = m$

$\phi(n) = 1$

$\phi(n-1) = m-1$

$$\underbrace{t(n)}_{O(m)} + \underbrace{\Phi(n)}_{1} - \underbrace{\Phi(n-1)}_{m-1} = O(1)$$

"Wenn ich viel arbeite, muss sich das Potential stark verringern"

$$\boxed{t(n) + \Phi(n) - \Phi(n-1)}$$

Assoziatives Array:

Paare an Tupeln $(k_1, v_1) \ldots (k_n, v_n)$,
wobei keys $k_1 \ldots k_n$ eindeutig sind.
Wir sagen v_i ist value von key k_i

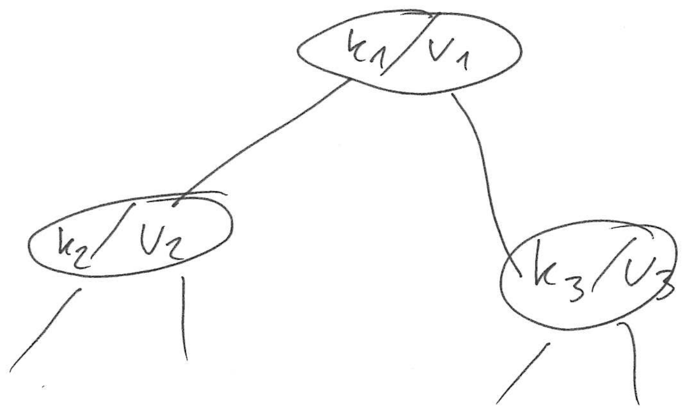
insert(k,v): Fügt neues Tupel (k,v) ein.
Fehler, falls Duplikat eingefügt wird (key)

search(k): Liefert value von k. oder
$\perp$ falls es nicht vorhanden ist.

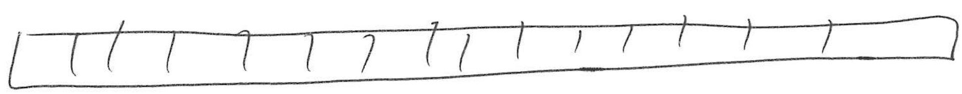
delete(k): Lösche Tupel mit key k.

edit (k,v): ersetze Value von key k mit v.

Binärbaum



Ordnung auf Keys:
z.B. lexikogr.
auf Bit Darstellung.

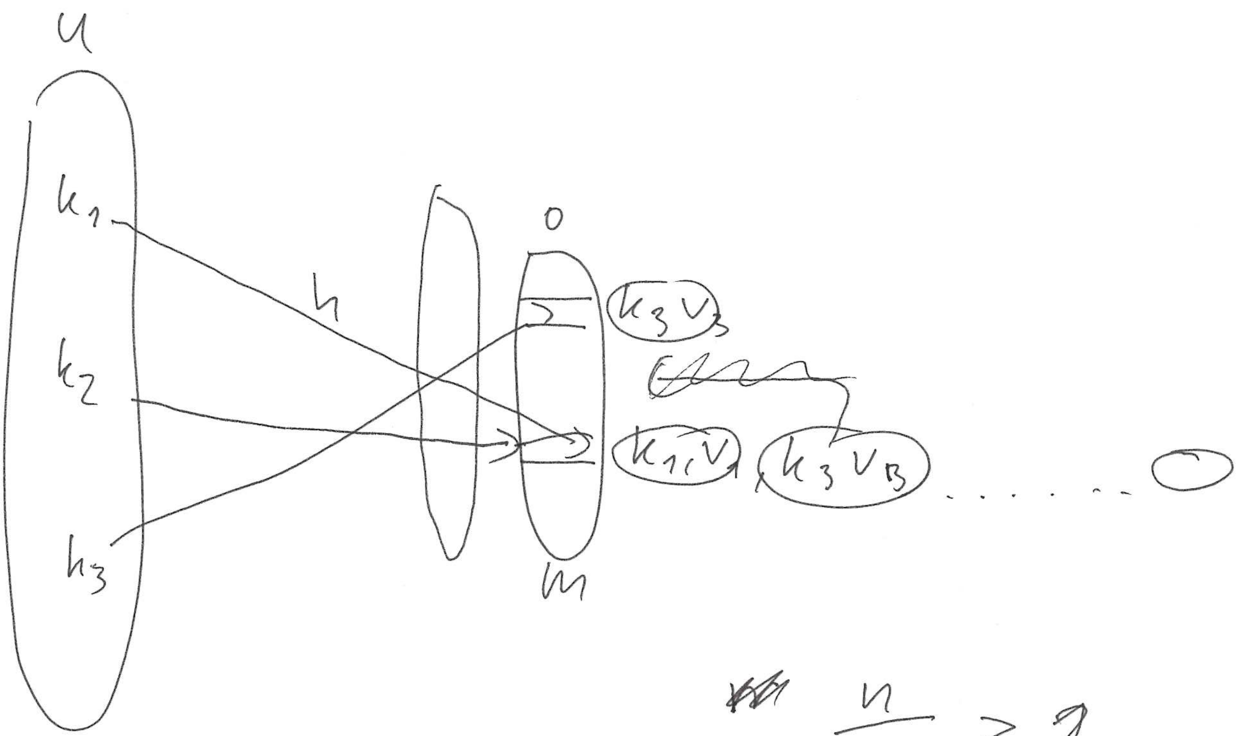


4

Füge keys in AVL / Splay-Baum ein und speichere bei jedem Key den Value.

$$\left.\begin{array}{l}\text{insert}\\ \text{search}\\ \text{delete}\\ \text{edit}\end{array}\right\}\ O(\log(n))$$

Hash Tabelle



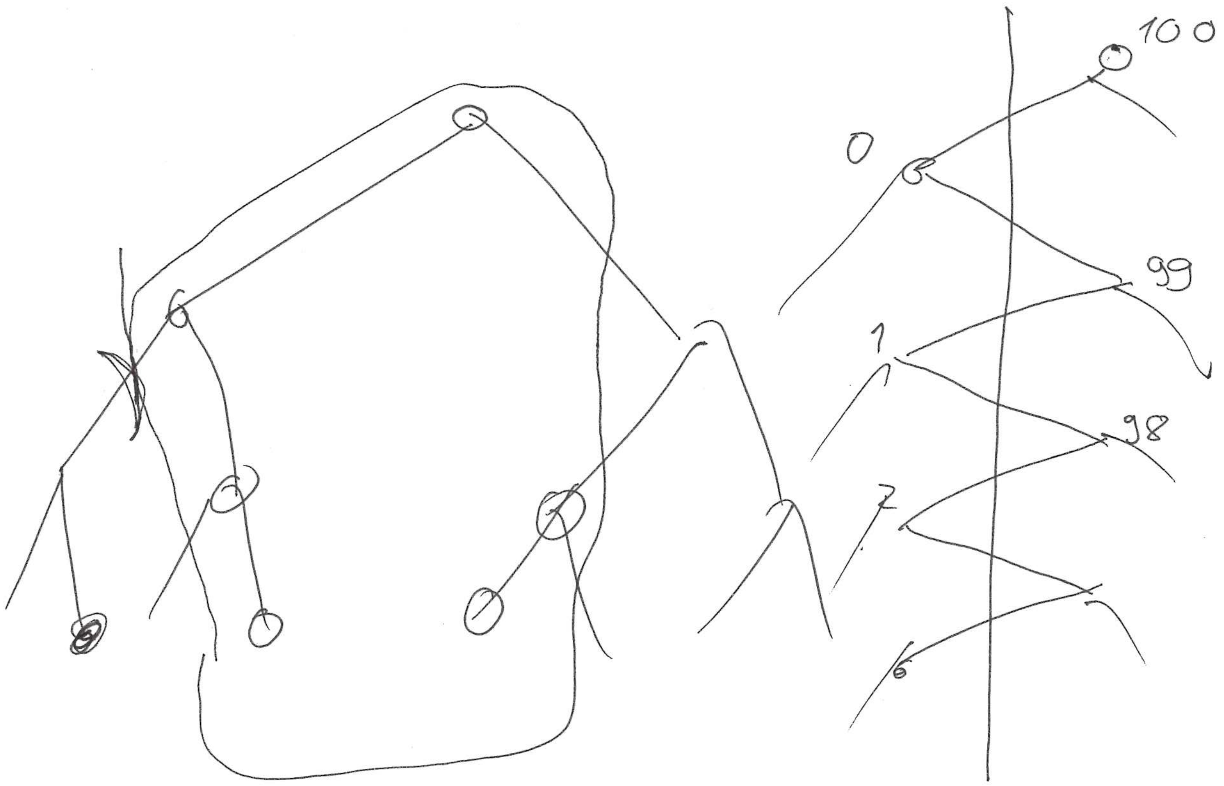
$$\frac{n}{m} > 1$$

Bäume (AVL / splay ...) HashMap (5)

$O(\log(n))^*$ Operationen pro Anweisung

$O(1)^*$ Operationen

getRange(l, r) liefert neues Array mit allen Keys zwischen l und r.



6