

# Algorithmen und Datenstrukturen (Master)

## WiSe 19/20

Benedikt Lüken-Winkels

November 27, 2019

### Contents

<b>1 Übung</b>	<b>8</b>
<b>2 Allgemeines</b>	<b>9</b>
2.1 Einschub: Erwartungswerte . . . . .	9

## Wörterbuchproblem

Menge  $S$  mit  $n$  Schlüsseln aus einem Universum  $U$ . Operationen: INSERT (darauf achten, dass die Balance nicht verloren geht), DELETE, LOOKUP (Im Baum runterlaufen, bis das Element gefunden wurde)

### Situationen

1.  $U$  linear geordnet, also existiert ein  $\leq$ -Test  $\Rightarrow$  Suchbäume
2.  $U$  ist ein Intervall  $\{0, \dots, N-1\}$  der gesamten Zahlen  $\Rightarrow$  Hashing

### zu 1:

**Randomisierte Suchbäume** Idee: Benutze Zufallszahlen zur Balancierung eines binären Suchbaums

**Binärer Suchbaum (Knoten-Orientiert)** Schlüssel werden in den  $n$  Knoten eines binären Baums gespeichert, sodass im linken Unterbaum des Knotens mit Schlüssel  $x$  alle Schlüssel  $< x$  **und** im rechten Unterbaum alle  $> x$ . Balanciert  $\Rightarrow \text{Höhe}(T) \leq \log n$ . Degeneriert  $\Rightarrow \text{Höhe}(T) = O(n)$

### Definition: Randomized Search Tree (RST)

Sei  $S = \{x_1, \dots, x_n\}$  eine Menge von  $n$  Schlüsseln. Jedem  $x_i$  wird eine zusätzlich eine Zufallszahl (auch Priorität genannt)  $prio(x_i)$  zugeordnet.  $prio(x_i)$  sind gleichverteilte reelle Zufallszahlen  $\in [0, 1]$  (Implementierung wären int-Zahlen, zB 32-bit).

Ein RST für  $S$  ist ein binärer Suchbaum für die Paare  $(x_i, prio(x_i))$ ,  $1 \leq i \leq n$ , sodass

1. normaler Knoten-orientierter Suchbaum für die Schlüssel  $x_1, \dots, x_n$
2. Maximumsheap bzgl der Prioritäten. dh  $prio(v) \geq prio(u)$ , falls  $v$  Parent.  $((u,v)$  sind Knoten in einem Baum).  $\Rightarrow$  Wurzel enthält maximale Priorität.

**Existenz** durch Algorithmus zum Aufbau (rekursiv).

- Wurzel enthält  $(x_i, p_i)$  mit  $p_i = prio(x_i)$  maximal
- Linker Unterbaum: RST für  $\{(x_i, p_i) | x_j < x_i\}$
- Rechter Unterbaum: RST für  $\{(x_k, p_k) | x_k > x_i\}$

Beispiel:  $S = \{1, \dots, 10\}$

- Schreibe Tabelle mit Prioritäten und Werten.
- Teile die Tabelle beim Maximum und schreibe es in die Wurzel. Wiederhole, bis alle Elemente geschrieben.

$\Rightarrow$  Wenn sich die Prioritäten genauso oder umgekehrt, wie die Schlüssel verhalten, erhält man einen degenerierten Baum. (bzgl  $\leq$ ). zB  $prio(x_i) = x_i$ . Dieser Fall ist sehr unwahrscheinlich, wenn sich bei der Priorität um gleichverteilte Zufallszahlen handelt.

## Operationen

- Lookup(x): normale suche in binärem Baum. Kosten  $O(\text{Höhe}(T))$
- Insert(x): Füge einen neuen Knoten v als Blatt  $(x, \text{prio}(x))$  gemäß des Schlüssels in den binären Baum ein, wobei  $\text{prio}(x)$  neue Zufallszahl (kann die Prio-Ordnung zerstören). Dann: Rotiere v nach oben, bis die Heap-Eigenschaft gilt, also  $\text{prio}(v) \leq \text{prio}(\text{parent}(v))$ . Kosten:  $O(\# \text{Rotationen}) = O(\text{Höhe}(T))$ . Alternativ: normales einfügen in binären Baum in absteigender Reihenfolge der Prioritäten.
- DELETE(x): Sei v der Knoten mit Schlüssel x ( $v = \text{Lookup}(x)$ ). Kosten:  $O(\# \text{Rotationen}) = O(1 + |L| + |R|)$ 
  1. Rotiere v nach unten, bis v ein Blatt ist. R = linkes Rückgrat des rechten Unterbaums von v. L = rechtes Rückgrat des linken Unterbaums.
  2. Entferne das Blatt.
- Split(y)  $\rightarrow S_1 = \{x \in S | x \leq y\}, S_2 = \{x \in S | x \geq y\}$  (Teile den Baum, indem y mit maximaler Priorität zur Wurzel rotiert wird)
  1. Insert( $y + \epsilon$ ) mit Priorität  $\infty$
  2. Entferne die Wurzel
- Join( $T_1, T_2$ ):  $S \leftarrow S_1 \cup S_2$ .  $T_1$  RST für  $S_1$  und  $T_2$  RST für  $S_2$ 
  1. Konstruiere T (Füge y zwischen  $\text{Max}(S_1)$  und  $\text{Min}(S_2)$  ein. Voraussetzung:  $\text{Max}(S_1) < \text{Min}(S_2)$ )
  2. Lösche die Wurzel (Durch runterrotieren des eingefügten Knotens y)

## Analyse des RST

Wir analysieren die erwarteten Kosten einer Delete-Operation (Insert  $\rightarrow$  umgekehrtes Delete). Sei T ein RST für die Menge  $\{x_1, \dots, x_n\}$  mit  $x_1 < x_2 < \dots < x_n$  der durch Inserts aufgebaut wurde. Betrachte die Operation  $\text{Delete}(x_k)$  für eine  $k, 1 \leq k \leq n$ . Für einen Knoten  $x_k$  im Baum T mit Suchpfad  $P_k$ ,  $L_k$  rechtes Rückgrat von  $T_l$  und  $R_k$  linkes Rückgrat von  $T_r$ . Kosten  $O(|P_k| + |L_k| + |R_k|)$ . Wir schätzen die Erwartungswerte

### Lemma 1:

- a)  $E(|P_k|) = H_k + H_{n-k+1} - 1$

$$k\text{-te HarmonischeZahl} = H_k = \sum_{i=1}^k \frac{1}{i} \quad H_k \leq \ln(x) + 1$$

- b)  $E(|L_k|) = 1 - \frac{1}{k}$
- c)  $E(|R_k|) = 1 - \frac{1}{n-k+1}$

**Beweis** Betrachte eine Permutation  $\pi : [1..n] \rightarrow [1..n]$  (bijektive Abbildung), die die Schlüssel absteigend nach ihren Prio Werten sortiert. Dann gilt:

1. Jede Permutation  $\pi$  ist gleichwahrscheinlich (Wahrscheinlichkeit  $\frac{1}{n!}$ ), da die Prioritäten gleichverteilte Zufallszahlen sind.
2. Man erhält den selben binären Baum durch Einfügen der Schlüssel in einen unbalancierten Baum in der Reihenfolge, die  $\pi$  angibt.  $\rightarrow$  gleiches Verhalten, wie ein zufälliger binärer Baum.
3. Baum wächst nur an den Blättern.

Trick: arbeite ab jetzt mit zufälliger Permutation statt den Prioritäten.  $\rightarrow$  normaler Binärbaum mit zufälliger Einfügereihenfolge.

**Teil a) des Lemmas**  $P_k$  ist Suchpfad für Knoten  $x_k$ . Seien  $P'_k$  und  $P''_k$  Teilfolgen von  $P_k$  mit:  $\forall v \in P'_k, key(v) \leq x_k$  und  $\forall u \in P''_k, key(u) \geq x_k$ .

*Proof.* Beobachtungen:

1.  $|P_k| = |P'_k| + |P''_k| - 1$  ( $x_k$  in beiden Teilfolgen)
2.  $P'_k$  = Menge der Knoten  $v$  mit:
  - Wenn  $v$  eingefügt wird, gilt  $key(v)$  ist maximal mit  $key(v) \leq x_k$
3.  $P''_k$  = Menge der Knoten  $u$  mit:
  - Wenn  $u$  eingefügt wird, gilt  $key(u)$  ist minimal mit  $key(u) \geq x_k$

Wir zeigen

1.  $E(|P'_k|) = H_k$
2.  $E(|P''_k|) = H_{n-k+1}$

zu 1)  $K$  mögliche Kandidaten für  $P'_k \{x_1, \dots, x_k\}$ . Spiel: Ziehe zufällig Schlüssel aus  $\{x_1, \dots, x_k\}$ .  $E(|P'_k|)$  = Erwartungswert, wie oft ein Kandidat gezogen wird, der  $\geq$  als alle vorher gezogenen ist (neues Maximum).  $A^k = E(|P'_k|)$  (Spiel A)

$$A^k = \sum_{i=1}^k \frac{1}{k} \cdot (1 + A^{k-i})$$

Im Zug  $x_i$  schließt  $x_1 \dots x_i$  au. Dann gleiches Spiel mit  $K-i$  Kandidaten.

$$\begin{aligned} A^k &= \frac{1}{k} (k + \sum_{i=1}^k A^{k-i}) \\ &= 1 + \frac{1}{k} \sum_{i=1}^k A^{k-i} \end{aligned}$$

Wir zeigen durch Induktion über  $k$ , dass  $A^k = H_k$

**IA**

$$= 1 + \frac{1}{k} \sum_{i=0}^k H_i$$

Eigenschaften der harmonischen Zahlen:

1.

$$\sum_{i=0}^k H_i = k \cdot (H_k - 1)$$

2.

$$H_k \leq 1 + \ln k$$

aus 1) folgt:

$$\begin{aligned} A^k &= 1 + \frac{1}{k} \cdot k \cdot (H_k - 1) \\ &= 1 + H_k - 1 \\ &= H_k \end{aligned}$$

Der Erwartungswert ist gleich der k-ten Harmonischen Zahl.  $E(P'_k) = H_k$ . Abschätzung von  $E(P''_k) =: B^k$ . (Spiel B) kandidaten  $\{x_k \dots x_n\}$ : Zähle, wie oft ein neues Minimum gezogen wird. Dann sieht man leicht, dass

$$B^k = H_{n-k+1}$$

Beweis: symmetrisch.

$$\begin{aligned} E(|P_k|) &= E(|P'_k|) + E(|P''_k|) - 1 \\ &= A^k + B^k - 1 \\ &= H_k + H_{n-k+1} - 1 \end{aligned}$$

□

### Teil b) des Lemmas

*Proof.*  $L_k$  und  $R_k$  Seien  $L_k = v_1, \dots, v_l$  und  $R_k = u_1, \dots, u_m$

**Erwartungswerte** Spiel C: Ziehe zufällig Elemente aus  $\{x_1 \dots x_n\}$ . Sobald  $x_k$  gezogen wird: ??? Trigger ??? Sei  $C^k$  der Erwartungswert dieses Spiels, dh  $C^k = E(|L_k|)$ .

$$C^k = \frac{1}{k} \cdot A^{k-1} + \sum_{i=1}^{k-1} \frac{1}{k} \cdot C^{k-i}$$

im 1. Zug  $x_k$ , dass Spiel mit k-1 Kandidaten (alle kleiner, als  $x_k$ ).

$$C^k = \frac{1}{k} \cdot (H_{k-1} + \sum_{i=0}^{k-1} C^i)$$

Trick: Schätze die Differenz zweier aufeinanderfolgender  $C^i$ s =  $\delta_j = C^{j+1} - C^j$  ab.

$$\Rightarrow C^k = \sum_{j=1}^k \delta_j + C^0$$

Betrachte:

$$\begin{aligned} & (j+1) \cdot C^{j+1} - j \cdot C^j \\ &= (j+1) \frac{1}{j+1} (H_j + \sum_{i=0}^j C^i) - j \frac{1}{j} (H_{j-1} + \sum_{i=0}^{j-1} C^i) \\ &= H_j + \sum_{i=0}^j C^i - (H_{j-1} + \sum_{i=0}^{j-1} C^i) \\ &= H_j - H_{j-1} + C^j \\ &= \frac{1}{j} + C^j \end{aligned}$$

Wir wissen nun, dass

$$\begin{aligned} & (j+1) \cdot C^{j+1} - j \cdot C^j = \frac{1}{j} + C^j \\ & \frac{1}{j} = (j+1)C^{j+1} - (j+1) \cdot C^j \\ & \frac{1}{j(j+1)} = C^{j+1} - C^j = \delta_j \\ & \Rightarrow \delta_j = \frac{1}{j(j+1)} = \frac{1}{j} - \frac{1}{j+1} \\ & C^k = \sum_{j=1}^{k-1} \delta_j = \sum_{j=1}^{k-1} \left( \frac{1}{j} - \frac{1}{j+1} \right) \\ & = 1 - \frac{1}{k} \end{aligned}$$

□

**Teil c) des Lemmas** Spiel  $D^k = E(|R_k|)$ : Wie oft wird ein neues Minimum größer  $x_k$  gezogen, nachdem  $x_k$  gezogen wurde (Trigger).

*Proof.* symmetrisch:

$$D^k = \frac{1}{n-k+1} \cdot B^{k-1} + \sum_{i=1}^{k-1} \frac{1}{n+k-1} D^{i-k??}$$

...

$$D^k = 1 - \frac{1}{n-k+1}$$

□

**Satz** Sie T ein RST für eine Menge von n Schlüsseln. Dann gilt:

1. Die erwartete Laufzeit für Insert, Delete und Lookup ist  $O(\log n)$
2. Die erwartete Zahl der Rotationen bei Delete ist  $< 2$

**Beweis**

1. Kosten von Lookup =  $O(|P_k|)$ , Insert und Delete =  $O(|P_k| + |L_k| + |R_k|)$  Kosten:  
 $O(H_k + H_{n-k+1} + 1 - \frac{1}{k} + 1 - \frac{1}{n-k+1}) = O(H_n) = O(\ln n) = O(\log n)$
2. Erwartete Zahl der Rotationen:  $E(|L_k|) + E(|R_k|) < 2$

# 1 Übung

## Übung 3:

1) Durch entfernen von Kanten soll der Graph zerlegt werden. (Unions in umgekehrter Reihenfolge)

2) Zu zeigen:

$$a(z, n) \leq \lfloor \frac{4m}{n} \rfloor \text{ für } z = \alpha(m, n)$$

Definition von  $a$  und  $\alpha$

$$a(z, n) = \min\{j | A(z, j) > \log n\}$$

$$\alpha(m, n) = \min\{i | A(i, \lfloor \frac{4m}{n} \rfloor) > \log n\}$$

Behauptung:

$$a(\alpha(m, n), n) \leq \lfloor \frac{4m}{n} \rfloor$$

Beweis: indirekt. Annahme:

$$a(\alpha(m, n), n) > \lfloor \frac{4m}{n} \rfloor$$

$$\Rightarrow A(\alpha(m, n), \lfloor \frac{4m}{n} \rfloor) \leq \log n$$

Widerspruch zur Definition von  $\alpha$ , denn

$$A(\alpha(m, n), \lfloor \frac{4m}{n} \rfloor) > \log n$$

**3.a)** Union-Split-Find. (van Emde-Boas aht Datenstruktur mit  $\log \log n$  für Union-Split-Find. ) Gegeben ist eine Array

- Split(i): Markiere i
- Find(x): Finde nächste Markierung
- Union(x): Lösche Markierung x

Balancierter (blatt-orientierter) Baum zur Speicherung der markierten Elemente. Einfügen der markierten Elemente als Blätter des Baums

- Split = Insert
- Union = Delete
- Find = Locate

Platz = #Intervalle, Zeit  $O(\log n)$



### 3.b)

- Insert = Split
- Delete = Union
- FindMin = Find(1)

## 2 Allgemeines

### 2.1 Einschub: Erwartungswerte

Situation:  $n$  Ereignisse, die mit einer gewissen Wahrscheinlichkeit  $\text{prob}(i)$  auftreten. Jedes Ereignis besitzt einen Wert  $\text{val}(i)$ .

$$E(\text{val}) = \sum_{i=1}^n \text{prob}(i) \cdot \text{val}(i)$$

Spezialfall: Gleichverteilung:  $\text{prob}(i) = \frac{1}{n}$  für  $1 \leq i \leq n$ . Dann gilt:

$$E(\text{val}) = \frac{1}{n} \sum_{i=1}^n \text{val}(i) = \text{Mittelwert}$$