Algorithmen und Datenstrukturen (Master) WiSe 19/20

Benedikt Lüken-Winkels

December 5, 2019

Contents

Wörterbuchproblem

Menge S mit n Schlüssln 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 $\{0, ..., N-1\}$ 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 $\Rightarrow H\ddot{o}he(T) \leq logn$. Degeneriert $\Rightarrow H\ddot{o}he(T) = O(n)$

Definition: Randomized Search Tree (RST)

Sei $S = \{x_1, ..., x_n\}$ 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 eine binärer Suchbaum für die Paare $(x_i, prio(x_i)), 1 \leq i \leq n$, sodass

1. normaler Knoten-orientierter Suchbaum für die Schlüssel $x_i, ..., x_n$
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 einthält (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: $S = \{1, ..., 10\}$

- Schreibe Tabelle mit Prioriäten 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 degenrierten 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 $O(H\ddot{o}he(T))$
- Insert(x): Füge einen neuen Knoten v als Blatt $(x, prio(x))$ gemäß des Schlüssels in den binären Baum ein, wobei $prio(x)$ neue Zufallszahl (kann die Prio-Ordnung zerstören). Dann: Rotiere v nach oben, bis die Heap-Eigenschaft gilt, also $prio(v) \leq prio(parent(v))$. Kosten: O(#Rotationen) = O(Höhe(T)). Alternativ: normales einfügen in binären Baum in absteigender Reihenfolge der Prioritäten.
- DELETE(x): Sei v der knoten mit Schlüssel x (v = Lookup(x)). Kosten: O(#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 ∞
  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 Max(S_1) und Min(S_2) ein. Voraussetzung: Max(S_1) < 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). Seit T ein RST für die Menge $\{x_1, ..., x_n\} mit x_1 < x_2 < ... < x_n$ der durch Inserts aufgebaut wurde. Bertrachte die Operation 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 Rückgrad von T_l und R_k linkes Rückgrad von T_r. Kosten $O(|P_k| + |L_k| + |R_k|)$. Wir schätzen die Erwartungswerte

Lemma 1:

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

$$k - te\ HarmonischeZahl = H_k = \sum_{i_1}^{k} \frac{1}{i}\ H_k \leq ln(x) + 1$$

- 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 π ist gleichwahrscheinlich (Wahrscheinlichkeit $\frac{1}{n!}$), da die Prioritäten gleichverteilte Zufallszahlen sind.

2. Man erhält den selben binären Baum durh einfügen der Schlüssel in einen unbalancierten Baum in der Reihenfolge, die π angibt. $\rightarrow$ gleiches Vehalten, 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 $P'_k\{x_1, ..., x_k\}$. Spiel: Ziehe zufällig Schlüssel aus $\{x_1, ..., x_k\}$. E($|P'_k|$) = Erwartungswert, wie of 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 $x_1 ... x_i$ au. Dann gleiches Spiel mit K-i Kandidaten.

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

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. $$H_k \leq 1 + lnk$$

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 ist gleich der k-ten Harmonischen Zahl. $E(P_k') = H_k$. Abschätzung von $E(P_k'') =: B^k$. (Spiel B) kandidaten $\{x_k ... x_n\}$: Zähle, wie of 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 $L_k = v_1, ..., v_l$ und $R_k = u_1, ..., u_m$

Erwartungswerte Spiel C: Ziehe zufällig Elemente aus $\{x_1 ... x_n\}$. 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).

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

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

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

Beatrachte:

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

Wir wissen nun, dass

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

□

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 fpr Insert, Delete und Lookup ist O(logn)
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: $O(H_k + H_{n-k+1} + 1 - \frac{1}{k} + 1 - \frac{1}{n-k+1}) = O(H_n) = O(lnn) = O(logn)$
2. Erwartete Zahl der Rotationen: $E(|L_k|) + E(|R_k|) < 2$

1 Hashing

1.1 Hashing mit offener Adressierung

Tafel T $[0, ..., m-1], m \leq n, |S| = n$ Verwende die Folge von Hashfunktionen h_0, h_1

$$h_i(x) = f(x) + i \cdot g(x), i = 0, 1, ...$$

Häufig verwendet wird

$$h_i(x) = (x \cdot modm + i) \cdot modm$$

$\rightarrow$ Linear Probing.
Idee

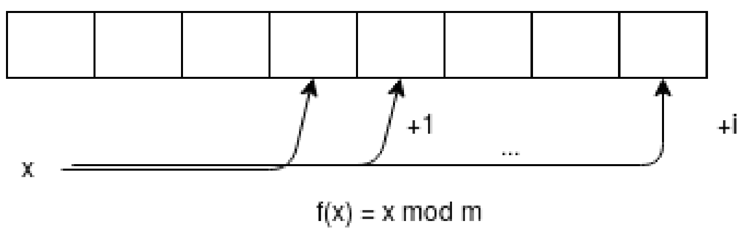

Figure 1: Idee: Offene Adressierungs-Tabelle

Operationen Tafelposition T[i] belegt oder frei.

- Init: alle frei
- Insert(x): betrachte die Tafelpositionen $T[h_0(x)], T[h_1(x)], T[h_2(x)]$ bis $T[h_i(x)]$ frei und speicher x dort ab. $T[h_i(x)] \leftarrow x$ und markiere $T[h_i(x)]$ als belegt. Voraussetzungen:
  1. $m \leq |S| = n$

2. $h_i(x)$ frei $i = 0, 1, 2, ...$ muss alle Tafelpoitionen durchlaufen

- Lookup(x): Teste die Tafelpoition $T[h_i(x)]$ für $i = 0, 1, 2, ...$ bis entweder $T[h_i(x)] = x$ erfolgreich oder $T[h_i(x)]$ ist frei. **Terminiert nicht**, wenn m=n und die Tafel voll ist und das gesuchte Element nicht vorhanden ist. Daher idR $m \geq n$
- Delete(x): (Idee 1):
  1. $j \leftarrow Lookup(x)$
  2. $T[j] \leftarrow frei$, dann sind auf j folgende Elemente nicht mehr erreichbar.

  (Idee 2):
  1. Dritter Zustand: 'gelöscht' (Details: Übung)

1.2 Perfektes Hashing

Situation: Statische Menge S von n Schlüsseln aus $[0, .., N-1]$. **Ziel:** Speichere S in einer Tafel der Größe O(n), sodass Lookup in Zeit O(1) realisiert werden kann (N » n, N sehr viel größer, als n). **Andere Formulierung:** Finde einer Hashfunktion $h : [0, ..., N-1] \rightarrow [0, ..., S-1]$ mit

1. S = Größe der Tafel und S = O(n)
2. h injektiv auf S

Zur Konstruktion oder Auswahl einer solchen Funktion Hashfunktion verwenden wir ein probabilitisches Verfahren (Zufallsverfahren).

Idee 2-stufiges Hashing-Schema (Hashing-Verfahren)

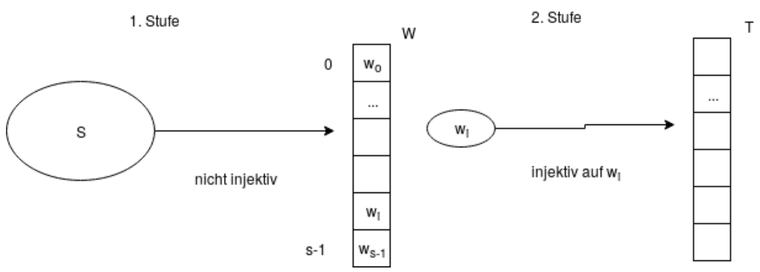

Figure 2: Idee: Perfekt Hashing Tabellen

Sei p eine Primzahl mit p > N und $S \in \mathbb{N}$ (Tafelgröße). Betrachte folgende Hashfunktionen:

$$h_k : [0, ..., N-1] \rightarrow [0, ..., S-1]$$

mit $h_k(x) = ((k \cdot x) mod p) mod s$ für alle $1 \leq k \leq p-1$ (Modulo Primzahl ergibt einen Restkörper, zB Inverses der Multiplikation). Diese Funktionen sind im Allgemeinen nicht injekt, dh h_k verteilt die Menge S auf s Buckets W_0^k, W_1^k, W_{s-1}^k.

$$\Rightarrow W_i^k = \{x \in S | h_k(x) = i\}$$

h_k injektiv auf $S \Leftrightarrow |W_i^k| \leq 1$ für $0 \leq i \leq s-1$

Lemma 1: Für jede Menge $S \subseteq \{0, ..., N-1\}, |S| = n$ gilt $\exists k, 1 \leq k \leq p-1 mit$

$$\sum_{i=0}^{s-1} \binom{|W_i^k|}{2} < \frac{n^2}{s}$$

(Anzahl der Kollisionen $< \frac{n^2}{s}$).

Beweis zunächst: Behauptung.

$$\sum_{k=0}^{p-1} \sum_{i=0}^{s-1} \binom{|W_i^k|}{2} < (p-1)\frac{n^2}{s}$$

Daraus folgt das Lemma (indirekt). Annahme, das Lemma 1 gilt nicht, dh $\forall 1 \leq k \leq p-1:$

$$\sum_{i=0}^{s-1} \binom{|W_i^k|}{2} \geq \frac{n^2}{s}$$

$$\Rightarrow \sum_{i=0}^{s-1} \binom{|W_i^k|}{2} \geq \frac{n^2}{s}$$

Widerspruch zur Behauptung!

Beweis zur Behauptung.

$$(1) \sum_{k=0}^{p-1} \sum_{i=0}^{s-1} \binom{|W_i^k|}{2}$$

= Anzahl der Paare $(k, \{x, y\}$, $x \neq y$ und $h_k(x) = h_k(y)$ (Anzahl der Kollisionen).
Wir schätzen zunächst den Betrag für 2 feste Werte $x \neq y$ zur Behauptung ab.

Sei also $x \neq y$:
(1) = Anzahl aller Ks mit $(k \cdot x \cdot mod p) mod s = (k \cdot y \cdot mod p) mod s$

$$\Leftrightarrow ((k \cdot x \cdot mod p) - (k \cdot y \cdot mod p)) mod s = 0$$

$$k \cdot x \cdot mod p - k \cdot y \cdot mod p = i \cdot s \text{ für ein } i \in \mathbb{Z}$$

$$k \cdot (x-y) mod p = i \cdot s$$

Es gibt maximal $\frac{2(p-1)}{s}$ mögliche Lösungen. Da p eine Primzahl ($\mathbb{Z}_p$ ist ein Körper) hat jede dieser Gleichungen höchstens 1 Lösung für k. $\Rightarrow$ Beitrag eines festen Paares $x \neq y$ zu (1) ist maximal $\frac{2(p-1)}{s}$

$$\begin{aligned} \Rightarrow (1) &\leq \binom{n}{2} \cdot \frac{2(p-1)}{s} \\ &= \frac{n(n-1)}{2} \cdot \frac{2(p-1)}{s} \\ &< \frac{n^2(p-1)}{s} \end{aligned}$$

□

Folgerung 1 (aus Lemma 1)
Für s=n (dh Tafel der Größe n):

$$\exists k, 1 \leq k \leq p-1 \text{ mit } \sum_{i=0}^{n-1} |W_i^k|^2 < 3n$$

Beweis für Folgerung 1. Betrachte Lemma 1 für s=n

$$\exists k : \sum_{i=0}^{n-1} \binom{|W_i^k|}{2} < n \tag{1}$$

$$\sum_{i=0}^{n-1} \frac{|W_i^k| \cdot (|W_i^k| - 1)}{2} < n \tag{2}$$

$$\sum_{i=0}^{n-1} |W_i^k| \cdot (|W_i^k| - 1) < 2n \tag{3}$$

$$\sum_{i=0}^{n-1} |W_i^k|^2 < 2n + \sum_{i=0}^{n-1} |W_i^k| (= n) \tag{4}$$

$$< 3n \tag{5}$$

□

Folgerung 2 Für $s = n^2$ dh quadratische Tafelgröße.
$\exists k' : 1 \leq k \leq p-1$, sodass die Hashfunktion

$$k_{k'} : x \rightarrow (k'x \cdot mod p) mod s$$

injektiv auf S ist dh $|W_i^k| \leq 1$ für $0 \leq i \leq s-1$.
$\Rightarrow$ Für Tafeln mit quadratische Größe existiert eine perfekte (injektive) Hashfunktion.

Beweis für Folgerung 2. Betrachte Lemma 1 mit $s = n^2$

$$
\begin{aligned}
\exists k' : \sum_{i=0}^{n^2-1} \binom{|W_i^k|}{2} &< 1 && (6)\\
\Rightarrow |W_i^k| &\leq 1 && (7)\\
\Rightarrow h_{k'} \text{ ist injektiv auf S} & && (8)
\end{aligned}
$$

(zu 6): dh Keine Kollisionen, bzw Doppeltbelegung in den W_i^k

□

Folgerung 2 zeigt Perfektes Hashing mit quadratischem Platz. Vermeidung des quadratischen Platzbedarfs durch ein 2-stufigen Hashing-Schema:

- Stufe 1: Wähle ein k gemäß Folgerung 1, dh Tafelgröße s=n und Hashfunktion h_k mit $\sum_{i=0}^{n-1} |W_i^k| < 3n$
- Stufe 2: Für jedes nicht-leere Bucket W_i^k der ersten Stufe verwende iene Tafel der Größe $s_i = |W_i^k|^2 (0 \leq i \leq n-1)$ und wähle ein k_i gemäß Folgerung 2, dh h_{k_i} ist injektiv auf W_i^k

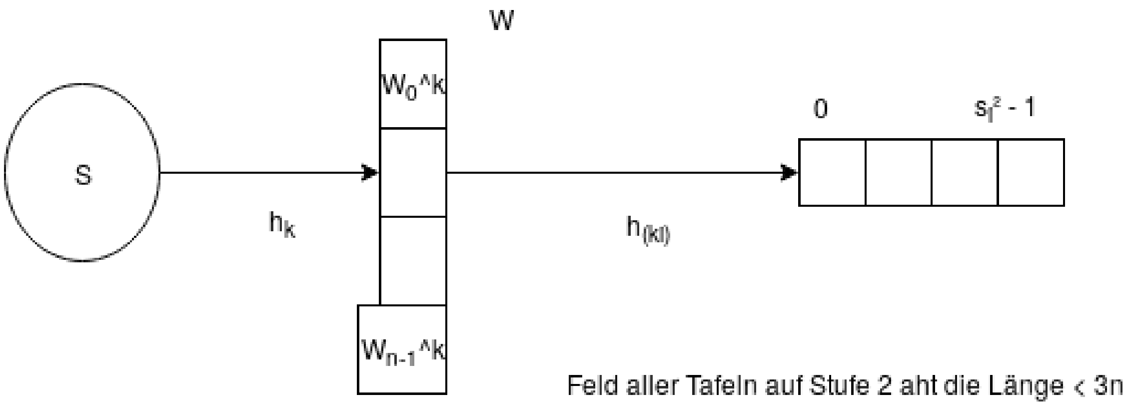

Figure 3: Konzept 2-Stufen Hashing

1.2.1 Datenstruktur

4 Felder + Variable k

- Variable k: (h_k der 1. Stufe)
- k[0,...,n-1]: k[i] ist k_i der 2. Stufe

- Size[0,...,n-1]: Size[i] = $|W_i^k|$ Bucketgrößen der 1. Stufe
- Ptr[0,...,n-1]: Pointer auf die Hashtafeln der 2. Stufe. Ptr[i] zeigt auf die Tafel B[0,...,Size[i]^2-1]
- T[0,...,3n]: Gesamtspeicherplatz aller B-Tafeln der 2. Stufe

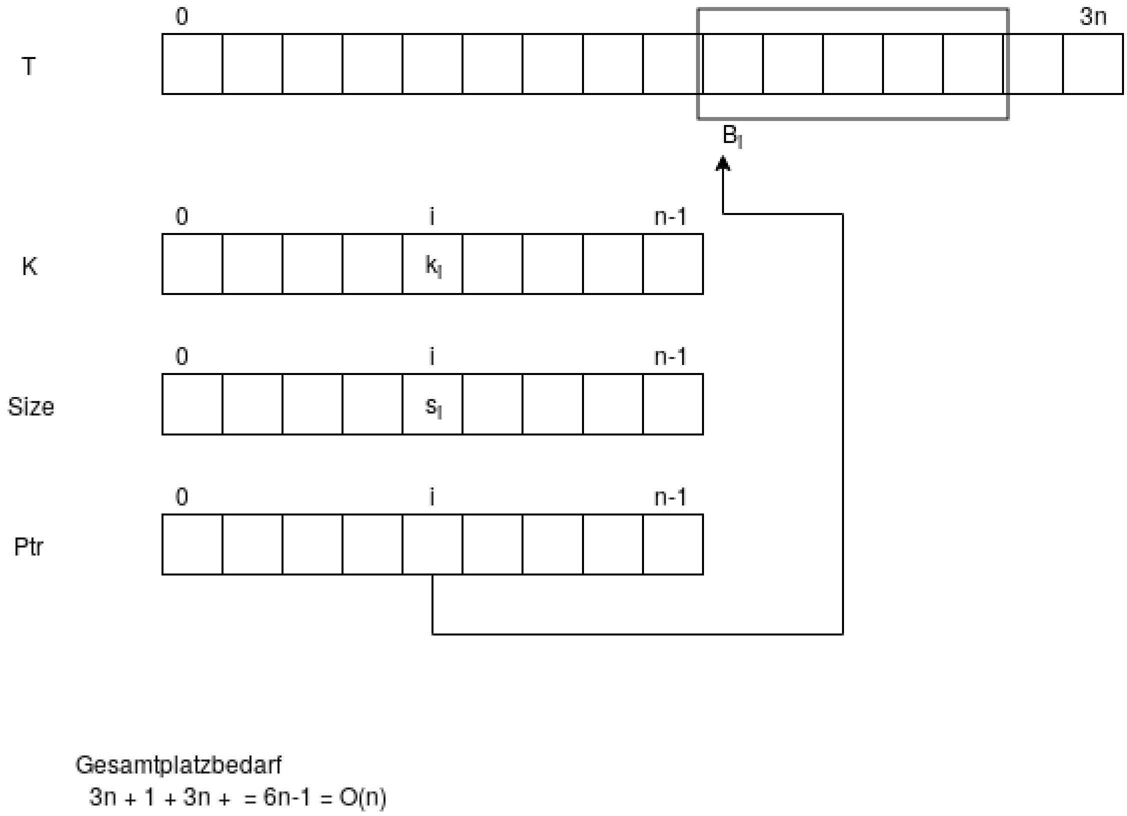

Figure 4: Datenstruktur: Perfekt Hashing

Abspeichern eines Elements x

$$i \leftarrow (k \cdot xmodp)modn \tag{9}$$
$$k' \leftarrow K[i] \tag{10}$$
$$s' \leftarrow Size[i] \tag{11}$$
$$j \leftarrow (k' \cdot xmodp)mods' \tag{12}$$
$$Ptr[i][j] \leftarrow x \tag{13}$$

2 Übung

Übung 3:

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

2) Zu zeigen:

$$a(z,n) \leq \lfloor \frac{4m}{n} \rfloor \ f\ddot{u}r \ z = \alpha(m,n)$$

Definition von a und α

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

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

Behauptung:

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

Beweis: indirekt. Annahme:

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

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

Widerspruch zur Definition von α , denn

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

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 Blätte rdes Baums

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

Platz = #Intervalle, Zeit O(logn)

3.b)

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

Übung 4:

1) Rekursive Funktion zum Aufbau eines RST
A[1,...,n] von Schlüsseln. P[1,...,n] Prioritäten

for $1 \leftarrow 0$ **to** n **do**
| $P[i] \leftarrow random()$
end

Rekursice Funktion RST(A,P,l,r) baut einen RST für A[l,...,r] und liefert Pointer auf die Wurzel.

```
class rst_node {
    key
    prio
    left,right: rst_node
}
```

def *RST(A,P,l,r)***:**
| **if** $l > r$ **then**
| | **return** null;
| $i \leftarrow max[P, l, r]$;
| $q \leftarrow$ new rstnode ;
| $q.key \leftarrow$ A[i] ;
| $q.prio \leftarrow$ P[i] ;
| $q.left \leftarrow$ RST(A,P,l,i-1) ;
| $q.right \leftarrow$ RST(A,P,i+1,r) ;
| **return** q;
end

2) Spiel B symmetrisch zu Spiel A.

3) Implementiertung von Hashing mit Verkettung. Idee: Tafelgröße s beliebig. Hashfunktion $h(x) = x \cdot mods$

def *Insert(x)***:**

end

def *Lookup(x)***:**

end

def *Delete(x)***:**

end

4) Belegungsfaktor $\beta = \frac{n}{m}$ m = Tafelgröße. Bei Hashing mit Verkettung ist $\beta =$ erwartete Länge einer Liste. Laufzeit für eine Operation $O(1+\beta) = O(1)$ für $\frac{1}{2} \leq \beta \leq 2$

Rehash Die Tabelle muss in eine größere Liste kopiert werden

def *Insert(x)***:**

... ;

$n \leftarrow n + 1$;

if $\frac{n}{m} > 2$ **then**

$m_0 \leftarrow m$;

$m \leftarrow 2m$;

$T' \leftarrow new\ int[m]$

Kopiere alte Tabelle in neue ;

end

def *Delete(x)***:**

$n \leftarrow n - 1$ **if** $\frac{n}{m} < \frac{1}{2}$ **then**

...;

$m \leftarrow \frac{m}{2}$;

Kopiere alte Tabelle in neue ;

end

3 Allgemeines

3.1 Einschub: Erwartungswerte

Situation: n Ereignisse, die mit einer gweissen Wahrscheinlichkeit prob(i) auftreten. Jedes Ereignis besitzt einen Wert val(i).

$$E(val) = \sum_{i_1}^{n} prob(i) \cdot val(i)$$

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

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