

Algorithmen und Datenstrukturen (Master)

WiSe 19/20

Benedikt Lücken-Winkels

January 8, 2020

Contents

1	Wörterbuchproblem	2
1.0.1	<u>zu 1:</u>	2
1.0.2	Definition: Randomized Search Tree (RST)	2
1.0.3	Operationen	3
1.0.4	Analyse des RST	3
1.0.5	Lemma 1:	3
2	Hashing	7
2.1	Hashing mit offener Adressierung	7
2.2	Perfektes Hashing	8
2.2.1	Datenstruktur	11
2.2.2	Aufbauzeit der Datenstruktur	13
2.2.3	Zusammenfassung	14
3	Planare Graphen	14
3.1	Definition	14
3.2	Planaritätstest	15
3.3	Wichtige Begriffe und Definitionen	15
3.3.1	Kontenzusammenhang	15
3.3.2	Beobachtung	16
4	Übung	17
5	Allgemeines	19
5.1	Einschub: Erwartungswerte	19
5.2	Integrierende Reihe	20

1 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

1.0.1 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 $\Rightarrow \text{Höhe}(T) = O(n)$

1.0.2 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.

1.0.3 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 ∞
 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)

1.0.4 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

1.0.5 Lemma 1:

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

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

- 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 durch Einfügen der Schlüssel in einen unbalancierten Baum in der Reihenfolge, die π 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 Schlüssel. 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: $E(|L_k|) + E(|R_k|) < 2$

2 Hashing

2.1 Hashing mit offener Adressierung

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

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

Häufig verwendet wird

$$h_i(x) = (x \cdot \text{mod } m + i) \cdot \text{mod } m$$

→ Linear Probing.

Idee

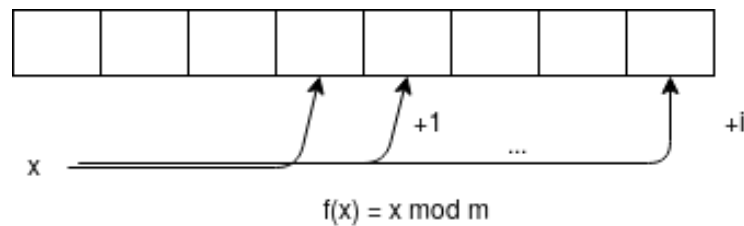


Figure 1: Idee: Offene Adressierungs-Tabelle

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 $i = 0, 1, 2, \dots$ muss alle Tafelpoitionen durchlaufen
- Lookup(x): Teste die Tafelpoition $T[h_i(x)]$ für $i = 0, 1, 2, \dots$ 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. $j \leftarrow \text{Lookup}(x)$
 2. $T[j] \leftarrow \text{frei}$, dann sind auf j folgende Elemente nicht mehr erreichbar.
 (Idee 2):
 1. Dritter Zustand: 'gelöscht' (Details: Übung)

2.2 Perfektes Hashing

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

1. $S = \text{Größe der Tafel}$ und $S = O(n)$
2. h injektiv auf S

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

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

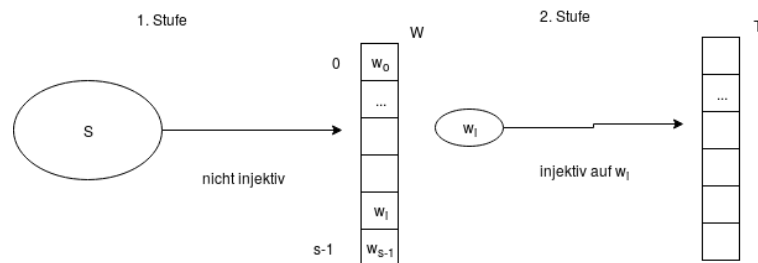


Figure 2: Idee: Perfekt Hashing Tabellen

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

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

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

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

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

Lemma 1: Für jede Menge $S \subseteq \{0, \dots, N-1\}$, $|S| = n$ 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 $\forall 1 \leq k \leq p-1$:

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

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 K s mit $(k \cdot x \cdot \bmod p) \bmod s = (k \cdot y \cdot \bmod p) \bmod s$

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

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

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

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 $s=n$

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

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

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

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

$$< 3n \quad (5)$$

□

Folgerung 2 Für $s = n^2$ dh quadratische Tafelgröße.

$\exists k' : 1 \leq k \leq p-1$, sodass die Hashfunktion

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

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 $s = n^2$

$$\exists k' : \sum_{i=0}^{n^2-1} \binom{|W_i^k|}{2} < 1 \quad (6)$$

$$\Rightarrow |W_i^k| \leq 1 \quad (7)$$

$$\Rightarrow h_{k'} \text{ ist injektiv auf } S \quad (8)$$

(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

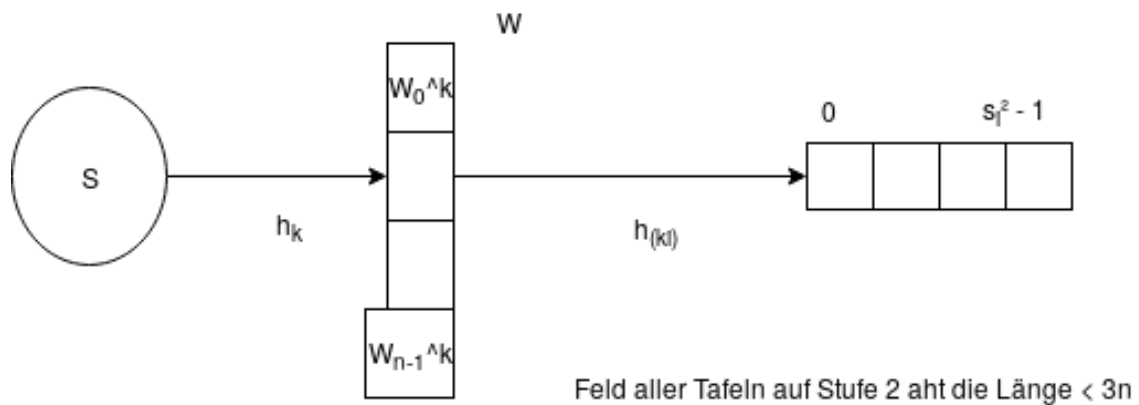


Figure 3: Konzept 2-Stufen Hashing

2.2.1 Datenstruktur

4 Felder + Variable k

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

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

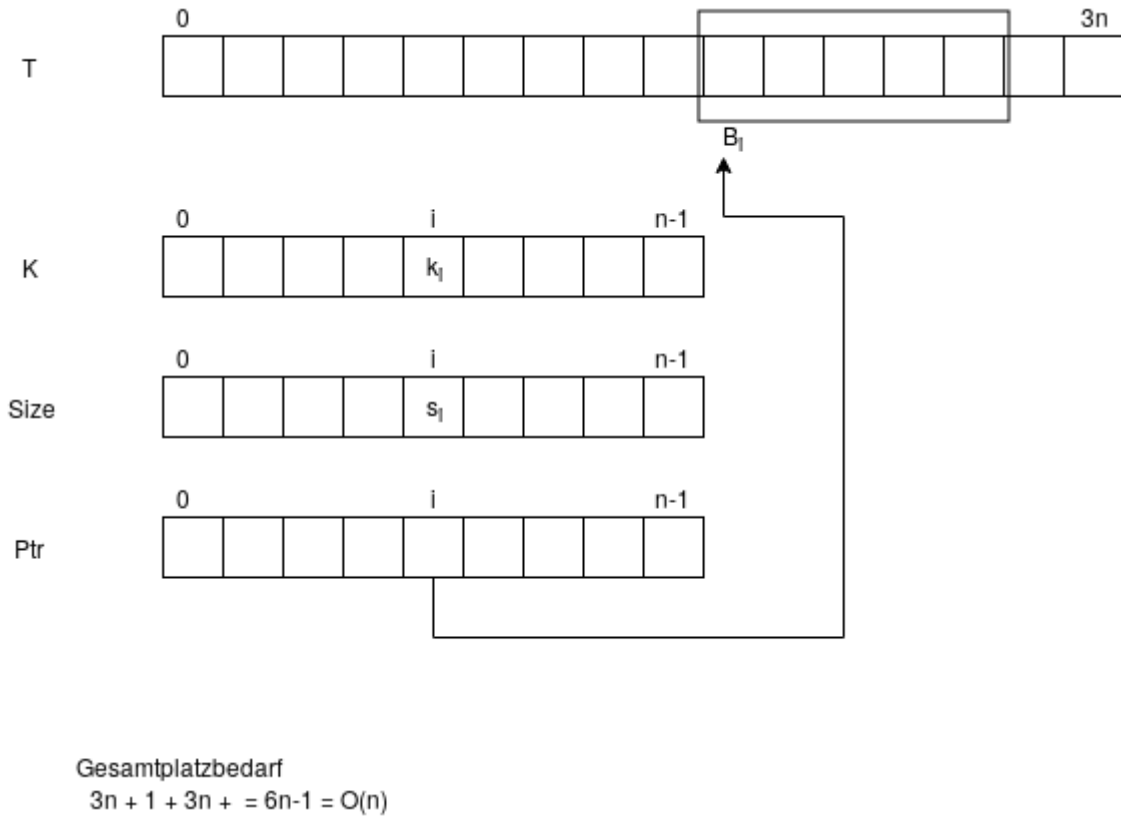


Figure 4: Datenstruktur: Perfekt Hashing

Abspeichern eines Elements x

$$i \leftarrow (k \cdot x \bmod p) \bmod n \quad (9)$$

$$k' \leftarrow K[i] \quad (10)$$

$$s' \leftarrow \text{Size}[i] \quad (11)$$

$$j \leftarrow (k' \cdot x \bmod p) \bmod s' \quad (12)$$

$$\text{Ptr}[i][j] \leftarrow x \quad (13)$$

2.2.2 Aufbauzeit der Datenstruktur

Wir finden man die k bzw k' Werte der ersten und zweiten Stufe (gemäß Folgerung 1 und 2). Aus Folgerungen 1 und 2 folgt: Es existiert immer mindestens ein Wert für k (dh eine geeignete Hashfunktion h_k). **Aufbaualgorithmus:** Teste alle k -Werte. Auf 1. und 2. Stufe

```
def Aufbau:
    for  $k = 1$  to  $p - 1$  do
        | 1.Stufe: Teste ob  $\sum_{i=0}^{n-1} |W_i^k|^2 < 3n$ ;
    end
end
```

Kostet im worst case $O(p \cdot n) = O(n \cdot N)$

2.Stufe für jedes Bucket W_i^k

```
def Aufbau:
    for  $k' = 1$  to  $n$  do
        | Teste ob  $h_{k'}$  injektiv auf  $W_i^k$ ;
    end
end
```

Kostet für $W_i^k O(p \cdot |W_i^k|)$ für alle Buckets $W_i^k (0 \leq i \leq n)$. Gesamtlaufzeit:

$$O\left(\sum_{i=0}^{n-1} p \cdot |W_i^k|\right) \quad (14)$$

Eine genauere Analyse zeigt, dass es viele k -Werte mit den geforderten Eigenschaften gibt.

Folgerung 3 (aus Lemma 1)

Für mindestens die Hälfte aller k , $0 \leq k \leq p - 1$, gilt

$$\sum_{i=0}^{n-1} |W_i^k|^2 < 5n (s = n)$$

Folgerung 4 (aus Lemma 1)

Für mindestens die Hälfte aller k' , $0 \leq k' \leq p - 1$, gilt

$$h_{k'} : x \rightarrow (k' \cdot x \bmod p) \bmod 2 \cdot n^2$$

ist injektiv auf S (angedwandt: $W_i^k s$. Beweis analog zum Beweis der Folgerung 1 und 2.

Änderungen in der Datenstruktur Auf 2.Stufe Tafelgrößen verdoppeln, dh jeweils Größe $2 \cdot |W_i^k|^2$. Platzbedarf der 2.Stufe:

$$\sum_{i=0}^{n-1} 2 \cdot |W_i^k|^2 = 2 \sum_{i=0}^{n-1} |W_i^k|^2 < 10n$$

Insgesamt: Platz $13n = O(n)$.

Aufbau Stufe 1: Wähle ein zufälliges $k \in \{1, \dots, p-1\}$ bis Folgerung 3 erfüllt. Wahrscheinlichkeit dafür ist jeweils $\geq \frac{1}{2}$. **Frage:** Wie hoch ist der Erwartungswert für die Anzahl der Tests. Analog zum Münzwurf: Wie viele Würfe, bis eine bestimmte Seite erscheint? Erwartungswert für diese Zahl (Integrierende Reihe):

$$\sum_{i=0}^{\infty} \frac{1}{2^i} \cdot i = \frac{0.5}{(1-0.5)^2} = 2$$

Erwartete Laufzeit für Stufe 1 ist dann $O(2 \cdot |S|) = O(n)$. Analog für Stufe 2: Erwartete Zahl der Tests = 2. \Rightarrow Gesamtlaufzeit $O(n)$.

2.2.3 Zusammenfassung

Man kann eine Menge S von n Schlüsseln aus $[0, \dots, N-1]$ so abspeichern, dass gilt

1. Platzbedarf ist $O(n)$
2. Erwartete Aufbauzeit $O(n)$
3. Zugriffszeit (Lookup) $O(1)$ worst-case

Dynamisierung ist möglich (Dynamic Perfect Hashing). Idee: Zeigen, dass die gewählten k -Werte mit großer Wahrscheinlichkeit für weitere Schlüssel funktionieren.

3 Planare Graphen

Literatur: Nishizekim und Chiba (Planar Graphs). Graph kann in die Ebene gezeichnet werden, ohne dass sich Kanten kreuzen. Wir betrachten ungerichtete Graphen $G=(V,E)$.

3.1 Definition

1. Ein Graph $G=(V,E)$ ist planar, wenn G eine planare Zeichnung hat.
2. Eine planare Zeichnung von G ordnet jedem Knoten v einen Punkt $p \in \mathbb{R}^2$ (Ebene) zu, $p = pos(x)$ heißt die Position von v , und jeder Kante (v, w) eine stetige Kurve im \mathbb{R}^2 mit den Endpunkten $pos(v)$ und $pos(w)$ zu, sodass sich diese Kurven paarweise nicht schneiden, außer in ihren Endpunkten.

G planar $\Leftrightarrow \exists$ planare Zeichnung.

Beispiele

nicht-planar $K_{3,3}$ bipartiter vollständiger Graph mit jeweils 3 Knoten auf jeder Seite. K_5 vollständiger Graph mit 5 Knoten. Ist ein Graph nicht planar, kann einer der beiden nicht-planaren Graphen gefunden werden.

3.2 Planaritätstest

Frage: ist der Graph planar?

1. Falls planar: planare Zeichnung
2. Falls nicht: möglichst kleiner nicht-planarer Teilgraph ($K_{3,3}$ oder K_5)

Beobachtung

- G ist planar $\Leftrightarrow \exists$ planare Zeichnung auf einer Kugeloberfläche
- Oberflächenstruktur von Polyedern kann durch planare Graphen dargestellt werden. (zB Würfel)

3.3 Wichtige Begriffe und Definitionen

3.3.1 Kontenzusammenhang

Der Zusammenhang eines Graphen (genauer Knotenzusammenhang).

- G heißt einfach zusammenhängend, wenn es für jedes Paar (v, w) von Knoten einen Pfad zwischen v und w gibt. Zusammenhangskomponenten sind zusammenhängende Teilgraphen.
- G heißt zweifach zusammenhängend (biconnected), falls $G \setminus \{v\}$ für jeden Knoten $v \in V$ zusammenhängend ist. Der Knoten, der den Graph zerfallen lässt, nennt man Artikulationspunkt (cut-vertex)
- Ein zweifach zusammenhängender Graph G heißt dreifach zusammenhängend, wenn für beliebige Knoten v, w gilt, $G \setminus \{v, w\}$ ist zusammenhängend. Die Knoten, die den Graph zerfallen lassen nennt man Separationspaar. **Splitgraphen** G_1 und G_2 durch Entfernung eines Separationspaares, erweitert um Kopieen des Entfertenen Paares. Kante innerhalb des Separationspaares kann existieren oder nicht.
- Allgemein k -fach zusammenhängend kann man beliebige $k-1$ Knoten entfernen, sodass G zusammenhängend bleibt.

1-fach, 2-fach und 3-fach kann mit DFS in Zeit $O(n+m)$ gelöst werden.

3.3.2 Beobachtung

Ein nicht-zweifach zusammenhängender Graph G ist genau dann planar, wenn seine zweifach Zusammenhangskomponenten (Blöcke) planar sind.

Idee Konstruiere für jeden Block eine planare Zeichnung, sodass alle Cut-Vertices außen liegen und kleben diese zusammen.

4 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 α

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

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

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 α , 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)

Übung 4:

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

```
for  $l \leftarrow 0$  to  $n$  do  
  |  $P[i] \leftarrow \text{random}()$   
end
```

Rekursive Funktion $\text{RST}(A, P, l, r)$ baut einen RST für $A[l, \dots, r]$ 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;  
     $i \leftarrow \max[P, l, r]$  ;  
     $q \leftarrow \text{new rstnode}$  ;  
     $q.\text{key} \leftarrow A[i]$  ;  
     $q.\text{prio} \leftarrow P[i]$  ;  
     $q.\text{left} \leftarrow \text{RST}(A, P, l, i-1)$  ;  
     $q.\text{right} \leftarrow \text{RST}(A, P, i+1, r)$  ;  
    return  $q$ ;  
end
```

2) Spiel B symmetrisch zu Spiel A.

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

```
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 β = 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
         $m_0 \leftarrow m$  ;
         $m \leftarrow 2m$  ;
         $T' \leftarrow \text{new int}[m]$ 
        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
```

5 Allgemeines

5.1 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:

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

5.2 Integrierende Reihe

$$x \leq 1$$

$$\sum_{i=0}^{\infty} x^i \cdot i = \frac{x}{(1-x)^2}$$