

Übungsblatt 4

Maurice Donner, Jan Hubrich, Adrian Müller

Aufgabe 1

Aufgabe 1 (Hashing, 4 + 1 + 2 + 1 Punkte)

Gegeben sei eine Hashtabelle mit 10 Buckets. Dabei sei die Hashfunktion h definiert als die Funktion, welche eine Zahl auf die Einerstelle abbildet, zum Beispiel gilt $h(13) = 3$.

- Verwenden Sie einerseits Hashing mit verketteten Listen, andererseits Hashing mit linearer Suche (zyklisches Array), um folgende Operationen durchzuführen:

Einfügen von 19, 52, 25, 29, 62, 12, 88, 10, 53

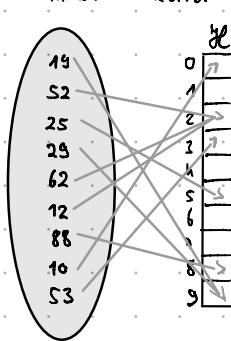
Geben Sie jeweils die Tabellen nach dem Einfügen von 62 und nach dem Einfügen von 53 an.

- Geben Sie die Anzahl der beim Einfügen betrachteten Hashtabellenplätze für beide Verfahren an!

- Nehmen Sie an, dass nach jedem Datum mit gleicher Wahrscheinlichkeit gesucht wird. Welche Kosten sind für das jeweilige Verfahren für eine erfolgreiche Suche zu erwarten?

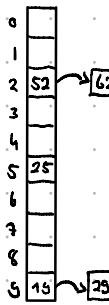
- Wie groß ist der Speicherverbrauch, wenn sowohl Zeiger als auch Element ein Maschinenwort benötigen, und mit einem Nullzeiger, das Ende einer Liste markiert werden kann?

1. Einfach verkettet

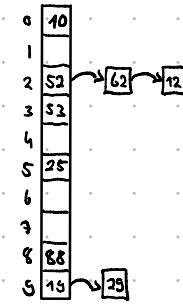


$$f(x) = x \% 10$$

Nach 62



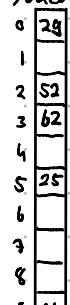
Nach 53



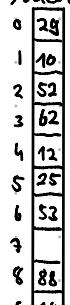
Linear Probing

$$f(x) = [x + g(i)] \% 10, \quad g(i) = i$$

Nach 62



Nach 53



2. • Mit einer einfach verketteten Liste brauchen wir eine Tabelle mit 13 Elementen. 4 Plätze bleiben dabei leer

• Bei Linear Probing brauchen wir lediglich 9 Plätze. Bei unserer Tabelle mit 10 Plätzen bleibt damit nur ein Platz frei.

3. Datum 19 52 25 29 62 12 88 10 53 Σ

Kosten E.V.L. 1 1 1 2 2 3 1 1 1 13

Kosten L.P. 1 1 1 2 2 3 1 2 4 17

Einfach verkettete Listen sind in diesem Fall also bei der Suche effizienter, verbrauchen jedoch mehr Speicherplatz

4. Einfach verkettete Liste: 28 $(13 + 15)$

Linear Probing 20 $(10 + 10)$

Aufgabe 2

Aufgabe 2 (Rechnungssystem, 6 + 2 Punkte)

Nehmen Sie an Sie haben eine große Datei die aus Tripeln der Form (*transaction, price, customerID*) besteht.

1. Entwickeln Sie einen Algorithmus, der den Gesamtzahlbetrag pro Kunden in erwarteter linearer Laufzeit berechnet.
2. Begründen Sie die erreichte Laufzeit!

1.

| | | | | | | | | | | | | | | |
|----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| ID | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
| Price | \$ | \$ | \$ | \$ | \$ | \$ | \$ | \$ | \$ | \$ | \$ | \$ | \$ | \$ |
| Customer | 0 | 0 | 1 | 0 | 2 | 0 | 2 | 1 | 3 | 3 | 3 | 4 | 5 | 1 |

16
000
...

Unsprüngliche Datei:

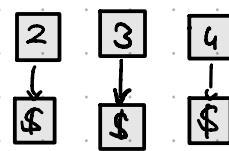


Hashtable



| | | | | | | | | |
|----------|----|----|----|----|----|----|----|----|
| Customer | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Sum | \$ | \$ | \$ | \$ | \$ | \$ | \$ | \$ |

...



1

Zähle Einträge in Datei, und initialisiere Hashtable mit verketteter Liste

2

Iteriere über transaction ID

- Jedes Mal, wenn eine neue Customer ID gefunden wurde, trage sie in Hashtabelle ein
 - ↳ Initialisiere Hashtabelleneintrag mit wert \$
- Jedes Mal, wenn eine customerID gefunden wurde, die sich bereits in der Hashtabelle befindet, addiere Preis

2.

→ Einträge in Datei Zählen: $\Theta(n)$

→ Über transaction ID iterieren $\Theta(n)$

↳ Suche nach CustomerID: $\Theta(1)$ (Da verkettete Liste)

↳ Miss: Initialisiere Preis $\Theta(1)$

↳ Hit: Addiere Preis $\Theta(1)$

Gesamt: $\Theta(n)$

Aufgabe 3

Aufgabe 3 (Entwurf einer Datenstruktur, 8 Punkte)

Gegeben sei ein Datentyp D mit Elementen der Form (*Schlüssel, Nutzdaten*). Die Größen von Schlüssel und Nutzdaten seien konstant. Die maximale Anzahl n der verschiedenen Schlüssel, die zu einem bestimmten Zeitpunkt in einem etwaigen System existieren, sei von vornherein bekannt. Die maximale Anzahl m der verschiedenen *Tupel* sei jedoch *nicht* von vornherein bekannt. Es können zudem auch Tupel mit dem gleichen Schlüssel vorkommen (im Extremfall können sogar alle vorhandenen Tupel den gleichen Schlüssel haben). Entwerfen Sie nun eine Datenstruktur für den Typ D , die folgendes leistet:

- Es existiert eine Operation $\text{Insert}(x_s, x_d)$, die ein Tupel (x_s, x_d) aus Schlüssel und Nutzdaten in die Datenstruktur einfügt. Ein Tupel wird dabei auch dann eingefügt, wenn in der Datenstruktur bereits ein Tupel mit dem selben Schlüssel existiert.
- Es existiert eine Operation $\text{Remove}(x_s)$, die für einen Schlüssel x_s ein Tupel mit passendem Schlüssel zurückgibt und aus der Datenstruktur entfernt. Falls kein solches Tupel in der Datenstruktur existiert, wird NIL zurückgegeben.
- Beide Operationen benötigen erwartet $O(1)$ Zeit.
- Die Datenstruktur hat einen Speicherbedarf von höchstens $O(\max\{n, m\})$.

Wir können ausnutzen, dass n bekannt ist:

◦ Initialisiere Hashtabelle (einfach verkettet) mit der Größe n

◦ Operation $\boxed{\text{Insert}(x_s, x_d)}$

1. Suche nach Tupel mit x_s $O(1)$

→ Falls noch kein Eintrag mit x_s existiert

- Füge neuen Eintrag mit x_s am Ende der Hashtabelle hinzu $O(1)$
- Erstelle Zeiger auf x_d $O(1)$

→ Falls bereits ein Eintrag mit x_s existiert

- Speicher Zeiger auf erstes Nutzdaten-Element in tmp $O(1)$
- Erstelle neues Element x_d und Zeiger darauf $O(1)$
- Verlege Zeiger tmp auf x_d $O(1)$

◦ Operation $\boxed{\text{Remove}(x_s)}$

1. Suche nach Tupel mit x_s $O(1)$

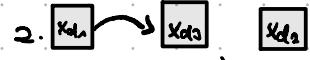
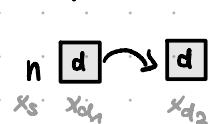
→ Falls nicht gefunden: NIL Zurückgeben

→ Falls x_s gefunden

- Sei D die Datenstruktur aus Tupel und Pointer: Gib $D.\text{next}$ zurück $O(1)$
- Speicher $D.\text{next}$ in tmp $O(1)$
- Verschiebe Zeiger vom nächsten Element nach vorne $O(1)$
- Lösche tmp $O(1)$

Speicherbedarf: Falls $n > m$: Die Tiefe der Hashtabelle ist 1
→ Es werden genau n Elemente gespeichert

Falls $m > n$: Es werden m Elemente gespeichert



$O(1)$

$O(1)$

$O(1)$

$O(1)$