Algorithmen und Datenstrukturen 1

ALGO1 · SoSe-2021 · tcs.uni-frankfurt.de/algo1

Wochenplan: Wörterbücher, Hashing

Revision 67cccce (2021-06-21)

Vorbereitung

Lies CLRS Kapitel 11 ohne 11.5 und schau das Video der Woche.

Dienstag

Aufgabe 1 (Von Hand laufen lassen und Eigenschaften).

a) (einfach) Füge die Schlüsselsequenz $7, 18, 2, 3, 14, 25, 1, 11, 12, 1332$ in eine Hashtabelle der Länge 11 ein, die verkettetes Hashing und die Hashfunktion $f(k) = k \mod 11$ benutzt.

b) (einfach) Füge die Schlüsselsequenz $2, 32, 43, 16, 77, 51, 1, 17, 42, 111$ in eine Hashtabelle der Länge 17 ein, die lineares Sondieren und die Hashfunktion $f(k) = k \mod 17$ benutzt.

c) Lösche 111 und 51 aus der in b) erzeugten Hashtabelle.

d) Angenommen, wir löschen ein Element x bei linearem Sondieren, ohne die Elemente im Cluster rechts von x wieder neu einzufügen. Gib eine kürzestmögliche Sequenz von Wörterbuchoperationen an, sodass diese modifizierte Variante zu einem falschen Ergebnis führt.

e) Sei S eine Sequenz von Schlüsseln, die in einer Hashtabelle A mittels verkettetem Hashing gespeichert sind. Gegeben A, ist es möglich den größten Schlüssel aus S effizient zu finden?

Aufgabe 2 (Teiler in der Divisionsmethode). Bei der *Divisionsmethode* wählen wir die Hashfunktion als $h(k) = k \bmod m$.

a) Betrachte die Hashfunktion $h(k) = k \bmod 10$ und die Schlüsselsequenz

$$K = 0, 5, 20, 40, 65, 15, 90, 95, 80, 55\,.$$

Warum ist diese Wahl der Hashfunktion problematisch für K?

b) Erkläre, warum wir für m eine Primzahl bevorzugen.

Aufgabe 3 (Faules Löschen bei linearem Sondieren). Die Methode aus 1d) hat nicht funktioniert, wir versuchen es also nochmal anders. Wenn wir ein Element an Position p löschen, dann hinterlassen wir jetzt eine Markierung, dass dort ein Element gelöscht worden ist.

a) Erkläre, wie `Search` und `Insert` modifiziert werden können, damit diese Methode funktioniert.

b) Erkläre welche Vor- und Nachteile diese Methode im Vergleich zu der Methode aus der Vorlesung hat.

Donnerstag

Aufgabe 4 (Spielserverstatistiken). Für dein neues, extrem erfolgreiches Onlinespiel *Hashnite* willst du ermitteln, ob die vielen gespielten Spielsitzungen von einer kleinen Gruppe extrem aktiver Spieler:innen kommt oder aus einer großen Gruppe verschiedener Spieler:innen, die unregelmäßig spielen. Jede Spieler:in hat eine eindeutige ID, und von deinem Spieleserver aus kannst du auf die Liste aller IDs aus allen vergangenen Spielesitzungen zugreifen.

a) Entwirf einen Algorithmus, der die Anzahl an *unterschiedlichen* Spieler:innen ermittelt, die jemals auf dem Server gespielt haben.
b) Entwirf einen Algorithmus, der die Spieler:in ermittelt, die die meisten Spielsitzungen gespielt hat.

Aufgabe 5 (Bitvektoren). Computer werden oft als *w-Bit Computer* bezeichnet, beispielsweise sind die meisten modernen Computer 64-Bit Computer. Das bedeutet, dass die Register und Speicherzellen jeweils w Bits speichern und primitive Datentypen wie Integer, Gleitkommazahlen und Zeiger mit w Bits dargestellt werden. Viele Programmiersprachen unterstützen Bit-Manipulationen mit konstantem Zeitaufwand, wie *bit shifts* (`<<` und `>>`) und bitweise logische Operationen (`|&^`). Wir wollen diese nutzen, um effizient Felder von Bits, also Elemente $B \in \{0,1\}^n$, als sogenannte *Bitvektoren* zu implementieren. Angenommen wir arbeiten auf einem w-Bit Computer. Löse die folgenden Teilaufgaben.

a) (einfach) Für $w = 8$, schreibe 2^4, `1<<4`, $2^8 - 1$, (2^7`^`2^4) und ($2^8 - 1$) `&` 2^4 in Binärdarstellung.
b) Zeige, wie ein Bitvektor B der Länge w kompakt dargestellt werden kann, sodass das i-te Bit in konstanter Zeit ausgelesen oder geflippt werden kann. (Die schlimmste Laufzeit darf also weder von n noch von w abhängen.)
c) Wie kann ein Bitvektor B der Länge n kompakt dargestellt werden, sodass das i-te Bit in konstanter Zeit ausgelesen oder geflippt werden kann? (Die schlimmste Laufzeit darf also weder von n noch von w abhängen.)
d) Entwirf eine Datenstruktur, die eine dynamische Menge $S \subseteq \{0, \dots, t\}$ darstellt und dabei die folgenden Operationen in konstanter Zeit unterstützt:
   - `insert`(a) fügt der Menge a hinzu, setzt also $S \leftarrow S \cup \{a\}$.
   - `remove`(a) löscht a aus der Menge, setzt also $S \leftarrow S \setminus \{a\}$.
   - `has`(a) liefert 1 wenn $a \in S$ und 0 sonst.

Aufgabe 6 (Sortieren in kleinen Universen, schwer). Sei $A[0 \dots n-1]$ ein Feld von Zahlen aus $\{0, \dots, n-1\}$. Entwirf einen Algorithmus, der A in Zeit $O(n)$ sortiert.

Aufgabe 7 (Nicht initialisierte Felder, sehr schwer). Wir wollen ein *riesiges* Feld A implementieren, dessen Einträge wir effizient auslesen und ändern können. Anfangs enthalten die Einträge von A „Müll“, und wegen der Größe des Feldes wollen wir beim Erzeugen des Feldes keine Zeit darauf verschwenden, alle Einträge zu initialisieren. Entwirf eine Datenstruktur, die linearen Platz in der Länge des Feldes benötigt, Auslesen und Ändern in erwarteter konstanter Zeit pro Eintrag unterstützt und nur konstante Zeit für die Initialisierung benötigt.