Datenstrukturen und Effiziente Algorithmen
Wintersemester 2020/21

Präsenzaufgaben 5

Aufgabe 1. (Qual der Wahl)
Welche Datenstrukturen bieten sich im Folgenden an, wenn wir effizient Suchen, Einfügen und Löschen wollen? Wie wähle ich die IDs (keys)?

1. Eine Anwendung (z.B. ein Spiel) erstellt dynamisch (d.h. bei Programmaufruf steht die Endanzahl der Objekte noch nicht fest) eine Reihe von Objekten und weist diesen eindeutige IDs zu. Der Einfachheit halber nehmen wir an, dass ein Objekt so lange bestehen bleibt, wie auch die Anwendung, dh. IDs werden nicht invalide.

2. Vorheriger Punkt, aber IDs können auch invalide werden, wenn Objekte nicht mehr benötigt und zerstört werden. Was ändert das womögich?

3. Eine Anwendung benötigt große Dateien (z.B. Bilder) an verschiedenen Stellen. Es kann vorkommen, dass eine Datei mehrfach angefordert wird. In diesem Fall soll sie nicht mehrfach geladen werden. Man könnte bei Anfrage prüfen, ob sie zuvor schon geladen wurde. (Dateipfade sind Strings)

4. Angenommen man möchte in einem sehr langen String (z.B. DNA Sequenz) für eine Menge von Teilstrings, alle Vorkommen dieser Teilstrings finden und merken. Später möchte man für einen gegeben Teilstring in konstanter erwarteter Zeit die Anzahl der Vorkommen abfragen (oder z.B. einen Zeiger auf den Kopf einer Liste aller Vorkommen).

Aufgabe 2. (Größenwahn)
Wir betrachten ein *riesiges* Datenfeld, das wir zur direkten Adressierung verwenden wollen. Initialisierung aller Einträge dauert uns zu lange, darum suchen wir eine Möglichkeit, zufälligen Datenmüll von einem tatsächlich gemachten Eintrag zu unterscheiden (idealerweise wollten wir alles mit NULL oder -1 initialisieren, um freie Felder anzuzeigen. Wenn wir das nicht machen, steht in jedem Feld ein zufälliger Zeiger). Beschreibe, wie man eine Datenstruktur erhält, in der die Operationen `Search`, `Insert` und `Delete` in Zeit $O(1)$ möglich sind. Jedes gespeicherte Objekt sollte nur $O(1)$ Speicher benötigen.

Hinweis: Benutze ein Hilfsarray, das als eine Art Stapel arbeitet und an dem man erkennt, ob ein Slot im riesigen Datenfeld belegt ist oder nicht. Seine Größe soll der Anzahl bereits eingefügter Objekte entsprechen.

Aufgabe 3. (Erwartete Anzahl an Kollisionen)
Sei h eine Hashfunktion, die n Schlüssel in ein Array der Länge m hasht. Es gelte die

simpe uniform hashing assumption. Was ist die erwartete Anzahl an Kollisionen, d.h. die erwartete Mächtigkeit von $\{\{k, l\} : k \neq l \wedge h(k) = h(l)\}$?

Aufgabe 4. (Überlauf)
Angenommen wir speichern n Schlüssel in einer Hashtabelle der Größe m mit Kollisionsauflösung durch Verkettung. Die Schlüssel stammen aus einer Menge U mit $|U| > nm$. Dann existiert eine Teilmenge von U der Größe n, s.d. alle Elemente auf denselben Slot hashen. (d.h. Zeit für Suche ist $\theta(n)$)