Lernzettel

Realisierung von Mengenstrukturen: HashSet, baum-basierte Mengen und Laufzeitanalysen

Universität: Technische Universität Berlin
Kurs/Modul: Algorithmen und Datenstrukturen
Erstellungsdatum: September 6, 2025

Lernzettel: Realisierung von Mengenstrukturen: HashSet, baum-basierte Mengen und Laufzeitanalysen

(1) Einordnung und Ziele. Mengenstrukturen speichern eindeutig Elemente und ermöglichen schnelle Mitgliedschaftsabfragen. Typische Realisierungen sind Hash-Tabellen (HashSet) und baum-basierte Mengen (balancierte Suchbäume). Die Laufzeitanalysen behandeln Kosten der Grundoperationen (enthält, einfügen, entfernen) und deren Abhängigkeiten von der Datenmenge. Ziel ist ein Verständnis der Vor- und Nachteile verschiedener Mengenimplementierungen sowie deren Zeit- und Speicherkomplexität.

(2) HashSet: Implementierung und Laufzeitanalysen. Die Grundidee eines HashSets ist eine Hashtable, in der jedes Element anhand einer Hash-Funktion $h : U \to \{0, \ldots, m-1\}$ einem Bucket zugeordnet wird:

$$\text{Bucket}(x) = h(x) \bmod m.$$

n = Anzahl der gespeicherten Elemente, m = Anzahl der Buckets, $\alpha = \dfrac{n}{m}$ (Auslastungsfaktor).

Es gibt zwei gängige Kollisionsauflösungsstrategien:

- Verkettung (Chaining): In jedem Bucket liegt eine Liste (oder eine andere dynamische Struktur) von Elementen. - Offene Adressierung (Open Addressing): Bei Kollision wird probeweise ein anderer Bucket geprüft (z. B. lineare Sondierung).

Für Verkettung gilt typischerweise:

$$T^{\text{avg}}_{\text{insert}} = O(1+\alpha), \quad T^{\text{avg}}_{\text{search}} = O(1+\alpha), \quad T^{\text{avg}}_{\text{delete}} = O(1+\alpha).$$

Damit ist bei einem konstanten Auslastungsfaktor α eine Erwartungslaufzeit von $O(1)$ pro Operation erreichbar.

Für offene Adressierung gelten in der Praxis grob:

$$T^{\text{avg}}_{\text{search}} = O\left(\frac{1}{1-\alpha}\right), \quad T^{\text{avg}}_{\text{insert}} = O\left(\frac{1}{1-\alpha}\right),$$

wobei α idealerweise klein gehalten wird.

Rehashing bzw. Neo-Allokation passiert, wenn das Ladenverhältnis eine Obergrenze überschreitet. Typisch wird m verdoppelt, und alle existierenden Elemente werden neu eingefügt.

$$\text{Amortisierte Kosten pro Operation} = O(1).$$

Wichtige Randbedingungen:

$$0 \le \alpha \le \alpha_{\max} \approx 0.7 \text{ bis } 0.9$$

je nach Implementierung und Speicherpolitik.

(3) Baum-basierte Mengen. Baum-basierte Mengen nutzen balancierte Suchbäume (z. B. AVL-Bäume oder Rot-Schwarz-Bäume), um sicherzustellen, dass die Höhe logarithmisch in der Anzahl der Elemente wächst.

$$h = O(\log n),$$

$$T_{\text{insert}} = O(h) = O(\log n), \quad T_{\text{search}} = O(h) = O(\log n), \quad T_{\text{delete}} = O(h) = O(\log n).$$

Balancierte Bäume garantieren Worst-Case-Laufzeiten, die logarithmisch zur Größe der Menge bleiben, während Hash-basierte Strukturen oft bessere konstante Faktoren haben, aber potenziell schlechtere Worst-Case-Verhalten.

(4) Laufzeitanalysen. - Worst-Case vs. Average-Case: Hashing kann im Worst-Case linear werden, balancierte Bäume bleiben logarithmisch. - Amortisierte Analysen: Kosten über mehrere Operationen summieren und/oder über Reallocationen verteilen. - Modellierung von Kosten: Jede Grundoperation hat Basiskosten, zusätzlich Kosten durch Rehashing oder Umstrukturierung.

Beispiel 1: Dynamische Arrays (Doubleing)

$$\text{Gesamtkosten beim Einfügen von } n \text{ Elementen} \leq 2n,$$

$$\Rightarrow \text{Amortisierte Kosten pro Einfügen} = O(1).$$

Beispiel 2: Hashing-Rehashing Bei Vergrößerung der Tabelle werden alle Elemente neu eingefügt:

$$T(n) = n + T_{\text{rest}} \Rightarrow \text{Amortisiert } O(1) \text{ pro Einfügen im Durchschnitt.}$$

(5) Praktische Hinweise und Vergleich. - HashSet bietet meist geringe Konstanten, kann aber im Worst-Case teurer werden. - Balancierte Mengen garantieren logarithmische Laufzeiten, haben aber oft höhere Konstantenfaktoren. - Speicherbedarf: Hashing benötigt zusätzlich Speicher für Buckets; balancierte Mengen benötigen Zeigerstrukturen.

(6) Übungsaufgaben. - Aufgabe 1: Beschreibe, wie sich der Auslastungsfaktor α auf die Laufzeit auswirkt. Gib Beispiele für verschiedene α. - Aufgabe 2: Vergleiche die Worst-Case-Laufzeit eines HashSets mit Verkettung vs. eines balancierten Baums beim Aufbau einer großen Menge. Diskutiere Speicherbedarf.