Testen und Dokumentieren

Aufgabenblatt 2

Torben Kunow (2316163)

Helena Lajevardi (2250233)

Malte Janssen (2271329)

23. Oktober 2017

Abstract

Ausarbeitung zu verschiedenen Implementierungen eines ADT SET, sowie deren Verifikation durch geeignete Testfälle und anschließendem Aufwandsvergleich.

I. Einführung

Die folgende Dokumentation beschreibt drei Implementations-Varianten eines ADT SET. Diese drei Varianten wurden auf Funktionalität getestet und vergleichend auf ihr Verhalten beim Entfernen eines Elements untersucht, sowie anschließend beurteilt.
Zur Umsetzung wurde in dieser Ausarbeitung die Programmiersprache Java [1] genutzt. Die, in ihrer Art, durchgeführten Untersuchungen können jedoch unabhängig der Programmiersprache nachvollzogen werden.

II. Implementierung von SET

Im Folgenden sind die Implementierungen von SET beschrieben. Als Basis für die Implementierungen wurde das in Abbildung. 1 dargestellte, vorgegebene Interface verwendet.

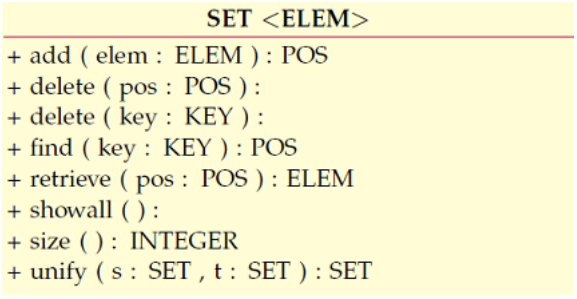

Abbildung 1: Interface SET

[1] https://en.wikipedia.org/wiki/Java_(programming_language)

I. Implementierung SetA

Bei dieser Implementation werden die Elemente der Menge in einem Array gespeichert. Reicht die Array-Größe beim Hinzufügen nicht aus, wird ein größeres Array allokiert. Beim Löschen eines Elements an einer Position werden die Elemente umkopiert (Siehe folgenden Code-Snippet).

II. Implementierung SetB

Diese Implementationsvariante speichert die Elemente in einem Array von Container-Klassen. Die Container-Klasse enthält je einen next- und previous-Index, sowie das eigentliche Element(Siehe Code-Snippet). Die Reihenfolge der Elemente wird durch den next- und previous-Index sortiert. Für den Abbruch der Suche wird ein Stop-Element verwendet. Wie in der vorhergehenden Variante, wird beim Hinzufügen eines Elementes ebenfalls ein größeres Array allokiert, bei nicht ausreichender Größe.

III. Implementierung SetC

Die dritte Implementation verwendet als Implementationstechnik eine einfach verkettete Liste mit zwei Dummy-Elementen und antizipativer Indizierung. Die Elemente der Liste sind in Container eingebettet. Die Container-Klasse enthält eine Referenz auf den nächsten Container. Für den Abbruch der Suche wird ebenfalls ein Stop-Element verwendet.

III. Verifikationstests der Implementierungsvarianten

Zur Verifikation der Funktionalität durch das bereitgestellte Interface wurde eine JUnit-Testklasse erstellt, die für alle drei Implementierungen die im Interface festgelegten Operationen testet. Folgende Tests wurden erfolgreich durchgeführt:

1. Testen von add
   (a) Hinzufügen von 5 Elementen. Ergebnis: Mit retrieve gefunden und size = 5.
   (b) Hinzufügen eines bereits vorhandenen Elementes. Ergebnis: size = 5
   (c) Hinzufügen von 1000 Elementen um das Vergrößern des Arrays zu erzwingen. Ergebnis: Mit retrieve gefunden und size = 1000.

2. Testen von delete
   (a) Löschen mit Key eines nicht vorhandenen Elementes. Ergebnis: size = 0.
   (b) Hinzufügen von 100 Elementen und Löschen des letzten Elementes mit Pos. Ergebnis: size = 99.
   (c) Löschen des ersten Elementes mit Pos. Ergebnis: size = 98.
   (d) Löschen von 98 Elementen mit Key. Ergebnis: size = 0.
   (e) Hinzufügen von 200 Elementen und Löschen mit Pos von 40 Elementen aus der Mitte. Ergebnis: size = 160.
3. Testen von find
   (a) Key eines vorhandenen Elementes wird übergeben. Ergebnis: Gültige Pos wird zurückgegeben.
   (b) Key eines nicht vorhandenen Elementes wird übergeben. Ergebnis: Ungültige Pos wird zurückgegeben.
4. Testen von retrieve
   (a) Hinzufügen eines Elementes. Ergebnis: Das Element wird mit retrieve zurückgegeben.
   (b) Eine nicht vorhandene Position wird übergeben. Ergebnis: Das zurückgelieferte Element ist null.
5. Testen von unify
   (a) Zwei leere Sets werden vereinigt. Ergebnis unifiedSet.size = 0.
   (b) Ein leeres Set e und ein Set s mit einem Element werden vereinigt. Ergebnis: e.size = 0, s.size = 1, unifiedSet.size = 1.
   (c) Ein Set mit 3 Elementen und eins mit 4 Elementen werden vereinigt, wovon ein Element in beiden enthalten ist. Ergebnis: unifiedSet.size = 6.

IV. Aufwandsanalyse

Hierbei wird das Verhalten der beiden Methoden delete(KEY) und delete(POS) der verschiedenen Implementationen untersucht. Dafür wird ein Zähler in die Methoden eingebaut, der die jeweiligen Operationen zählt. Getestet wird mit Listengrößen von 10^k, k= 1...5. Die folgenden Diagramme veranschaulichen das untersuchte Verhalten mit Blick auf die Position des Elementes, ob der Löschvorgang am Anfang, in der Mitte oder am Ende stattgefunden hat.

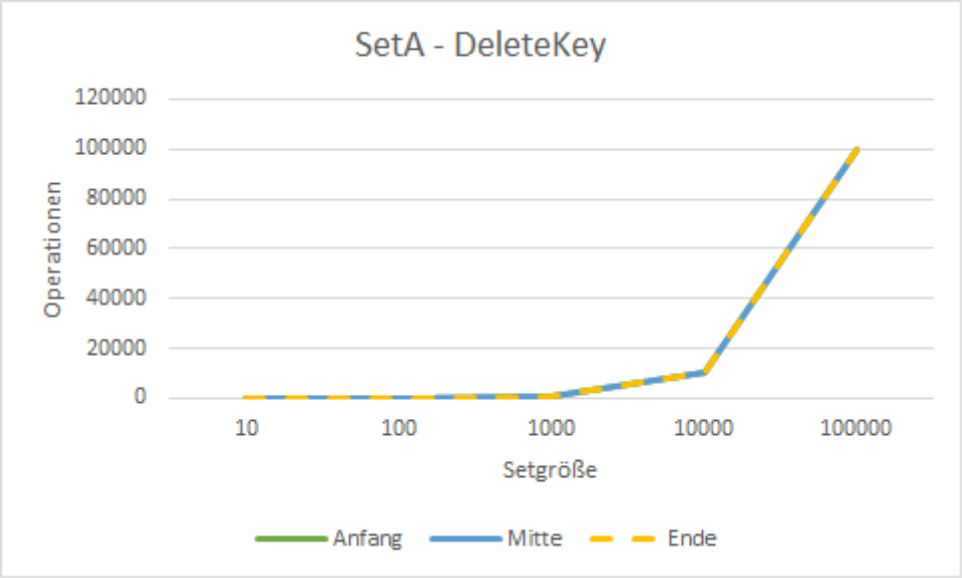

Abbildung 2: Löschvorgang mit Key in SetA

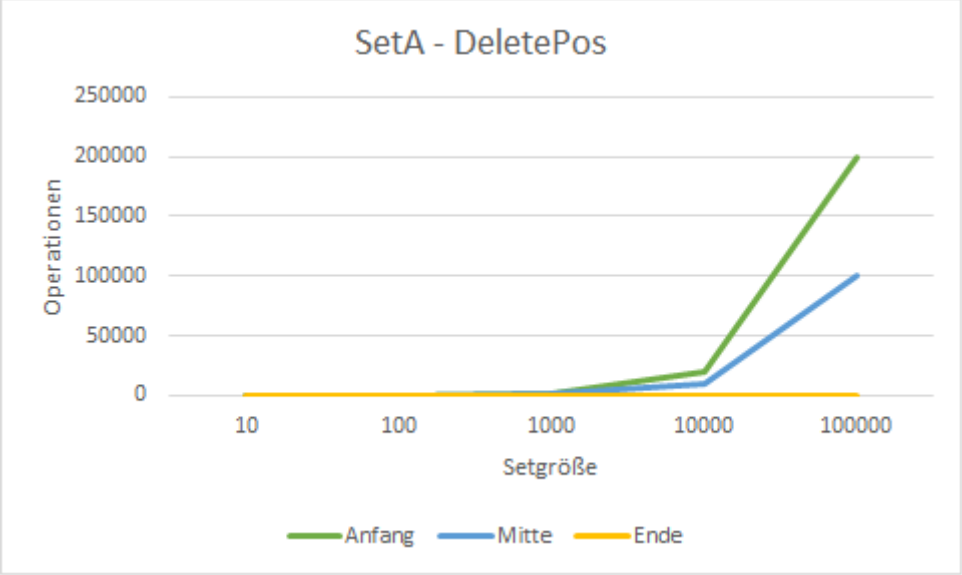

Abbildung 3: Löschvorgang mit Pos in SetA

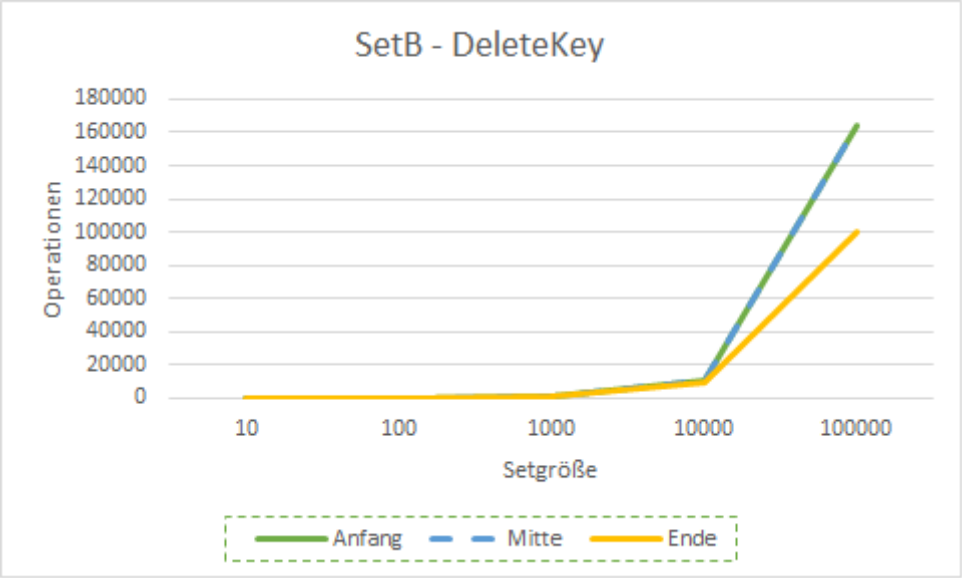

Abbildung 4: Löschvorgang mit Key in SetB

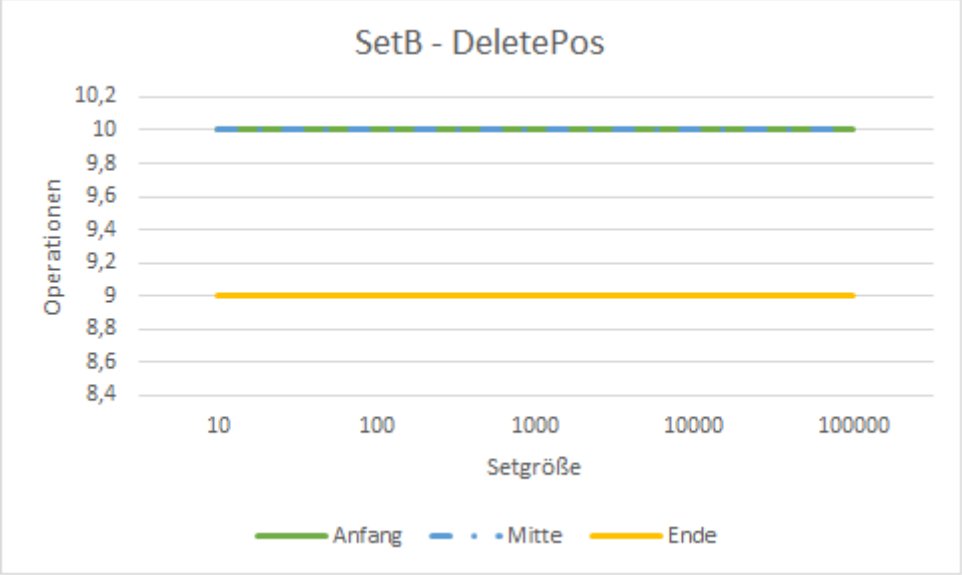

Abbildung 5: Löschvorgang mit Pos in SetB

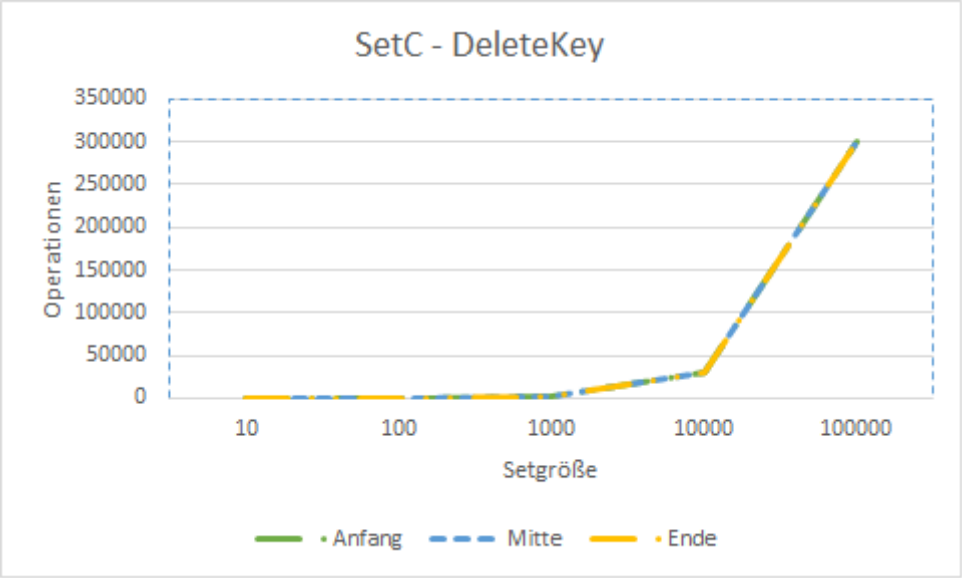

Abbildung 6: Löschvorgang mit Key in SetB

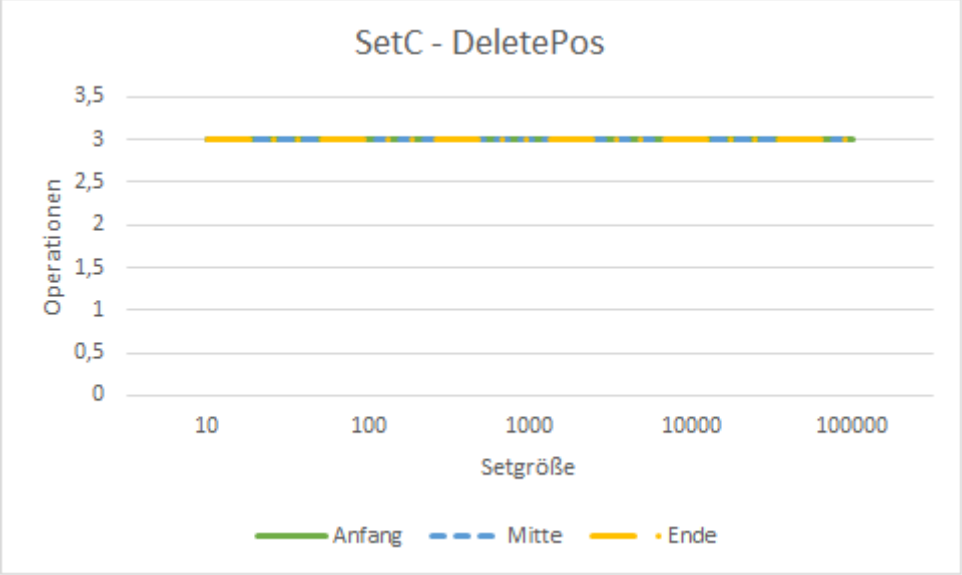

Abbildung 7: Löschvorgang mit Pos in SetC

V. Fazit

Schlussendlich war festzustellen, dass der Löschvorgang von SetA üblicherweise am längsten dauert, da das Zusammenrücken bei einer großen Anzahl von Elementen sehr aufwendig ist. Das Löschen mit Position am Anfang des Sets bei 100000 Elementen benötigt 200000 Operationen, löscht man hingegen das Element am Ende des Sets benötigt man nur 4. Bei SetB und SetC ist der Aufwand konstant und bei SetC am geringsten. Beim Löschen eines Elements mit Key ist jedoch SetA am schnellsten.