

Bericht Datenstruktur Programmieren 2 FHNW WIBB 2.61 2024

Aufbau

Im Rahmen dieser Untersuchung wurden die Datenstrukturen ArrayList, LinkedList, TreeSet und HashSet hinsichtlich ihrer Leistung beim Hinzufügen und Suchen von Objekten verglichen. Ziel war es, die Eignung der jeweiligen Struktur für unterschiedliche Anwendungsszenarien zu bewerten. Dazu wurden Tests durchgeführt, bei denen die Laufzeit für verschiedene Operationen und Datenmengen gemessen wurde.

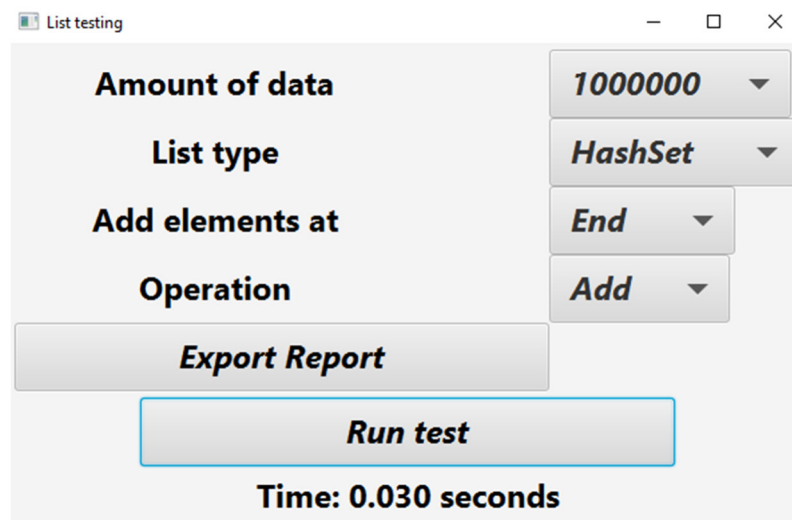
Methodik

Die Tests wurden mit einer Java-Anwendung durchgeführt, die in einer grafischen Benutzeroberfläche konfiguriert werden konnte. Anschliessend hat die Applikation jeweils pro Einstellung die Zeit gemessen und nach Durchspielen aller Optionen diese in einer .csv-Datei gespeichert. Meine Bemessungen basieren auf diesen Zeiten.

Die spezifischen Schritte waren:

1. Generierung von Datenobjekten (SampleData).
2. Auswahl der Datenstruktur (ArrayList, LinkedList, TreeSet, HashSet).
3. Durchführung der Operationen „Add“ und „Search“.
4. Messung der Laufzeit mit einer Auflösung von Millisekunden.

Beispiel einer Eingabe:



The screenshot shows a window titled "List testing" with the following elements:

- Amount of data:** A dropdown menu set to "1000000".
- List type:** A dropdown menu set to "HashSet".
- Add elements at:** A dropdown menu set to "End".
- Operation:** A dropdown menu set to "Add".
- Export Report:** A button.
- Run test:** A button with a blue border.
- Time: 0.030 seconds:** A text label at the bottom.

Abbildung 1 - Beispiel einer Parameterauswahl

Die Tests wurden für folgende Datenmengen durchgeführt: 10'000, 30'000, 100'000, 300'000 und 1'000'000 Elemente. Die Ergebnisse wurden in diesem Bericht zusammengefasst und grafisch dargestellt.

Ergebnisse

Die Diagramme unten zeigen die Zeit in Sekunden (y-Achse) in Abhängigkeit von der Datenmenge (x-Achse) für die Operationen "Add" (Hinzufügen von den Datenmengen entweder zu Beginn, in der Mitte oder am Ende der jeweiligen Liste) und "Search" (Sucht nach einem zufällig ausgewählten Objekt in einer Datenmenge von 1'000'000):

Add:

Die Diagramme zeigen, dass ArrayList nur beim Hinzufügen in der Mitte der Liste schneller ist, da **ArrayList** die jeweils neu hinzugefügten Objekte erst nach dem Verschieben der alten Objekte hinzufügen kann (konstant **$O(1)$**). Die doppelte Verkettung hilft der LinkedList beim Hinzufügen am Ende und am Anfang einer Liste deutlich schneller zu sein als eine ArrayList. Besonders zu erwähnen ist hier klar, dass das Hinzufügen von Daten in der Mitte der Liste (bei **LinkedList**) enorm viel Zeit benötigt. Dies ist auf die lineare Zeitkomplexität (**$O(n)$**) bei der Suche zurückzuführen. Wenn eine solche Funktion benötigt wird, sollte immer auf eine LinkedList verzichtet werden.

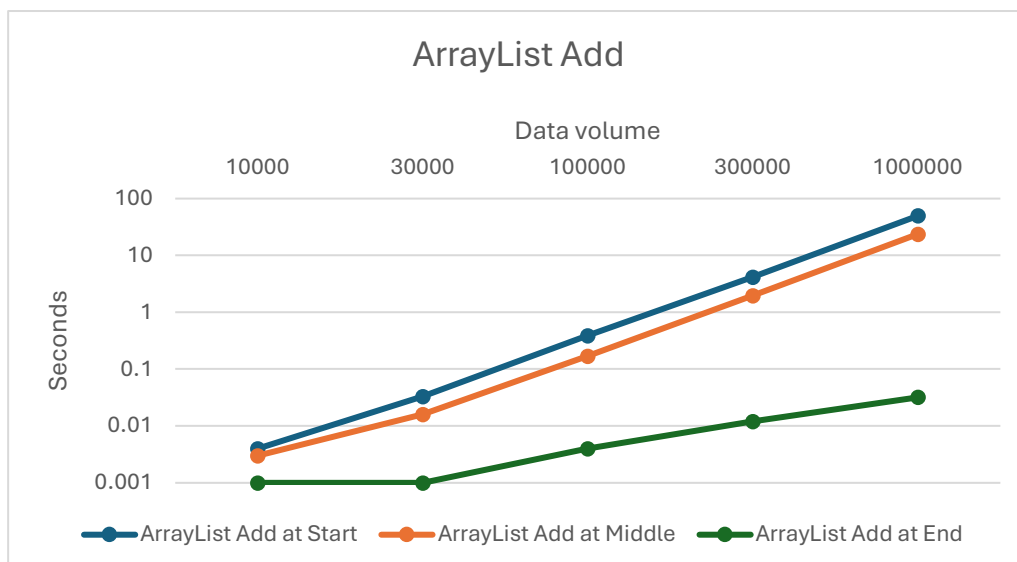


Abbildung 2 - ArrayList Add

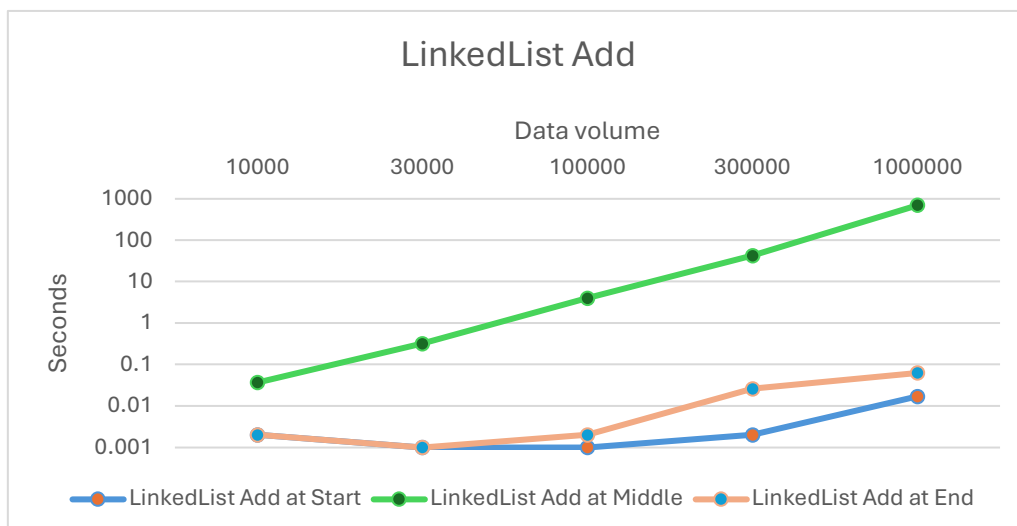


Abbildung 3 - LinkedList Add

Das **HashSet** zeigte eine nahezu konstante Zeit beim Hinzufügen von Daten, unabhängig von der Datenmenge. Diese Effizienz ist der Hashing-Strategie geschuldet, die in der Regel eine Zeitkomplexität von **$O(1)$** aufweist. Das **TreeSet** wies eine erwartungsgemäss konsistente Leistung auf, da es auf einer balancierten Baumstruktur basiert. Die Zeitkomplexität von **$O(\log n)$** führte zu einem leichten, aber stetigen Anstieg der Laufzeit mit zunehmender Datenmenge. Dies macht TreeSet zu einer guten Wahl für Szenarien, in denen eine sortierte Datenstruktur benötigt wird.

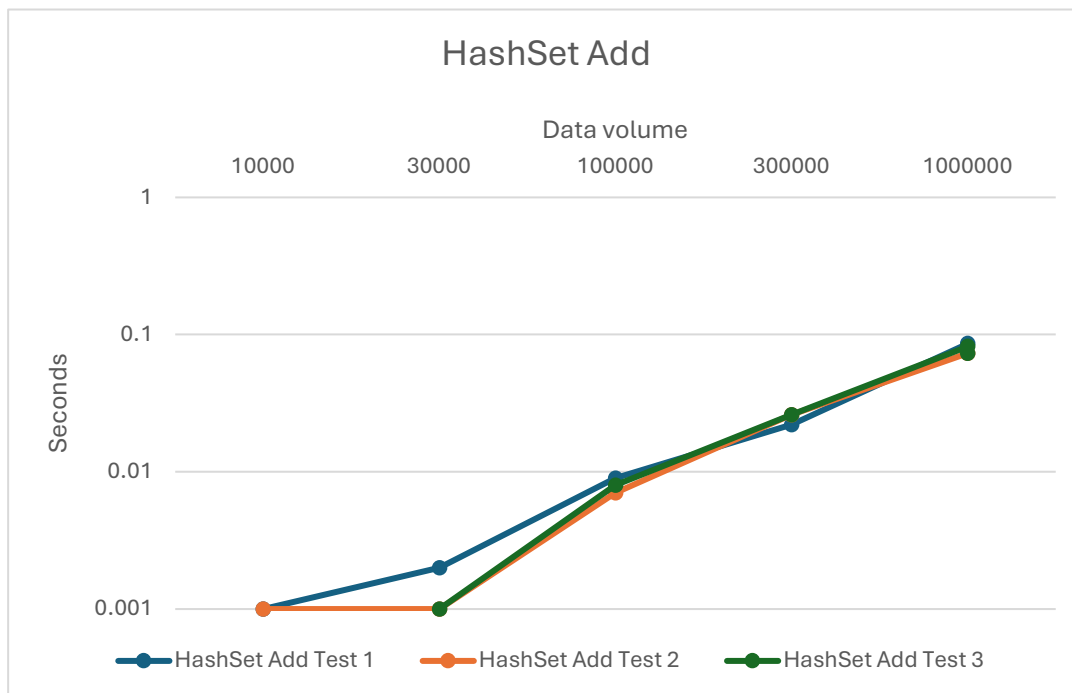


Abbildung 4 - HashSet Add

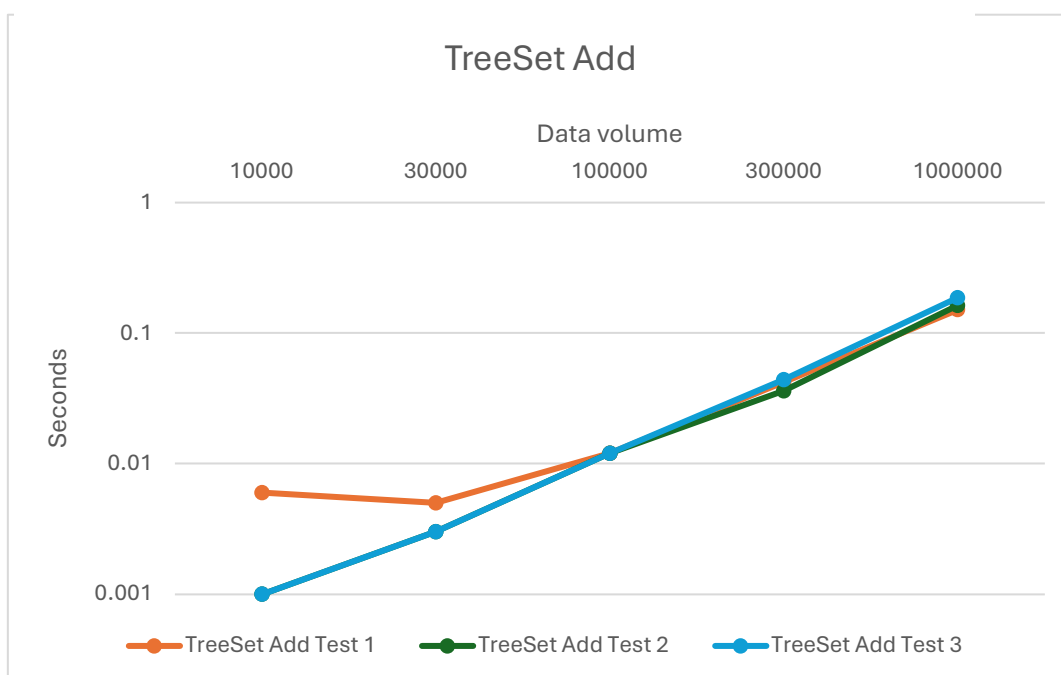


Abbildung 5 - TreeSet Add

Search:

Die Tabelle zeigt die durchschnittliche Zeit, die benötigt wird, um ein Objekt in den jeweiligen Datenstrukturen zu finden. Die Tests wurden mit einer Datenmenge von 1.000.000 Elementen durchgeführt. Dabei wurden fünf Durchläufe pro Datenstruktur gemacht, und die Ergebnisse sind wie folgt zu interpretieren:

ArrayList: Die Suchzeit in einer ArrayList steigt mit der Anzahl der Elemente, da eine lineare Suche (Zeitkomplexität $O(n)$) durchgeführt wird. Dies wird durch die leicht variierenden Zeiten in der Tabelle bestätigt.

LinkedList: Ähnlich wie bei der ArrayList zeigt die LinkedList ebenfalls eine lineare Suche. Die Suchzeit ist vergleichbar mit der ArrayList, jedoch kann der Zugriff auf Knoten aufgrund der Verkettung etwas langsamer sein.

TreeSet: Das TreeSet zeigt eine konstante Suchzeit von 0 Sekunden (nicht ganz 0 aber halt nicht messbar in meinem Test) in der Tabelle, was auf die hohe Effizienz der balancierten Baumstruktur zurückzuführen ist.

HashSet: Das HashSet liefert ebenfalls konstante Zeiten von 0 Sekunden (nicht ganz 0 aber halt nicht messbar in meinem Test). Dies spiegelt die Zeitkomplexität von $O(1)$ wider, da die Suche über eine direkte Adressierung im Hash-Bucket erfolgt.

List Type	Operation	Data Amount	Time (s)	Average
ArrayList	Search	1000000	0.012	0.0072
ArrayList	Search	1000000	0.002	
ArrayList	Search	1000000	0.007	
ArrayList	Search	1000000	0.009	
ArrayList	Search	1000000	0.006	
LinkedList	Search	1000000	0.012	0.0058
LinkedList	Search	1000000	0.002	
LinkedList	Search	1000000	0.002	
LinkedList	Search	1000000	0.007	
LinkedList	Search	1000000	0.006	
TreeSet	Search	1000000	0	0
TreeSet	Search	1000000	0	
TreeSet	Search	1000000	0	
TreeSet	Search	1000000	0	
TreeSet	Search	1000000	0	
HashSet	Search	1000000	0	0
HashSet	Search	1000000	0	
HashSet	Search	1000000	0	
HashSet	Search	1000000	0	
HashSet	Search	1000000	0	

Abbildung 6 - Tabelle zu Search

Die Tabelle verdeutlicht, dass HashSet und TreeSet bei der Suche am effizientesten sind, während ArrayList und LinkedList bei grossen Datenmengen aufgrund der linearen Suche weniger geeignet sind.

Fazit

Diese Untersuchung zeigt deutlich, wie wichtig die Wahl der richtigen Datenstruktur ist. **ArrayList** und **LinkedList** haben ihre Stärken, stossen jedoch bei grossen Datenmengen schnell an ihre Grenzen. Sie sollten deshalb nur gezielt eingesetzt werden.

HashSet und **TreeSet** hingegen überzeugen durch ihre Effizienz, vor allem beim Suchen/Search. Allerdings darf auch hier die Implementierung nicht vernachlässigt werden und vor der Implementierung stets geprüft werden und es ist vor der Implementierung immer zu prüfen, welche spezifischen Anforderungen an die Datenstruktur gestellt werden.