Vergleich von Datenstrukturen

ArrayList, LinkedList, TreeSet und HashSet

Autor: Loris Trifoglio

Inhalt

1 Einleitung

In diesem Bericht werden die vier Datenstrukturen ArrayList, LinkedList, TreeSet und HashSet verglichen. Die Datenstrukturen wurden mit den Operationen Add und Search getestet. Die Datenmenge wurde mit 1.000, 3.000, 10.000, 30.000, 100.000, 300.000, 1.000.000 und 3.000.000 Elementen variiert.

2 Datenstrukturen

2.1 ArrayList

Die ArrayList ist eine dynamische Liste, die Elemente in einem Array speichert. Die Elemente werden in der Reihenfolge ihrer Einfügung gespeichert.

2.2 LinkedList

Die LinkedList ist eine doppelt verkettete Liste, die Elemente in einer Kette von Knoten speichert. Die Elemente können in beliebiger Reihenfolge hinzugefügt, entfernt oder durchsucht werden.

2.3 TreeSet

Das TreeSet ist ein geordnetes Set, das Elemente in einem Baum speichert. Die Elemente werden in aufsteigender Reihenfolge gespeichert.

2.4 HashSet

Das HashSet ist ein ungeordnetes Set, das Elemente mithilfe eines Hash-Algorithmus speichert. Die Elemente werden in einer nicht bestimmten Reihenfolge gespeichert.

2.5 Beispiel Datensatz

Ein Datensatz besteht aus folgenden Elementen:

```
public SampleData() {
    this.ID = nextID++;
    this.name = generateName();
    this.number = (int) (Math.random() * 1000);
    this.dataNumber = Math.random();
    this.dataString = generateRandomString();
}
```

Beispiel: [607 | Seth Rose Rogers | 878 | 0.04669868868946281 | kG4ehaBeCG nxr6y1F2Xg Hxezx42CsC jHqzbjDJbS 2G4d4ZqsSz]

1. Operationen

2.6 Add

Die Add Operation fügt eine gewisse Anzahl Elemente in die Datenstruktur ein.

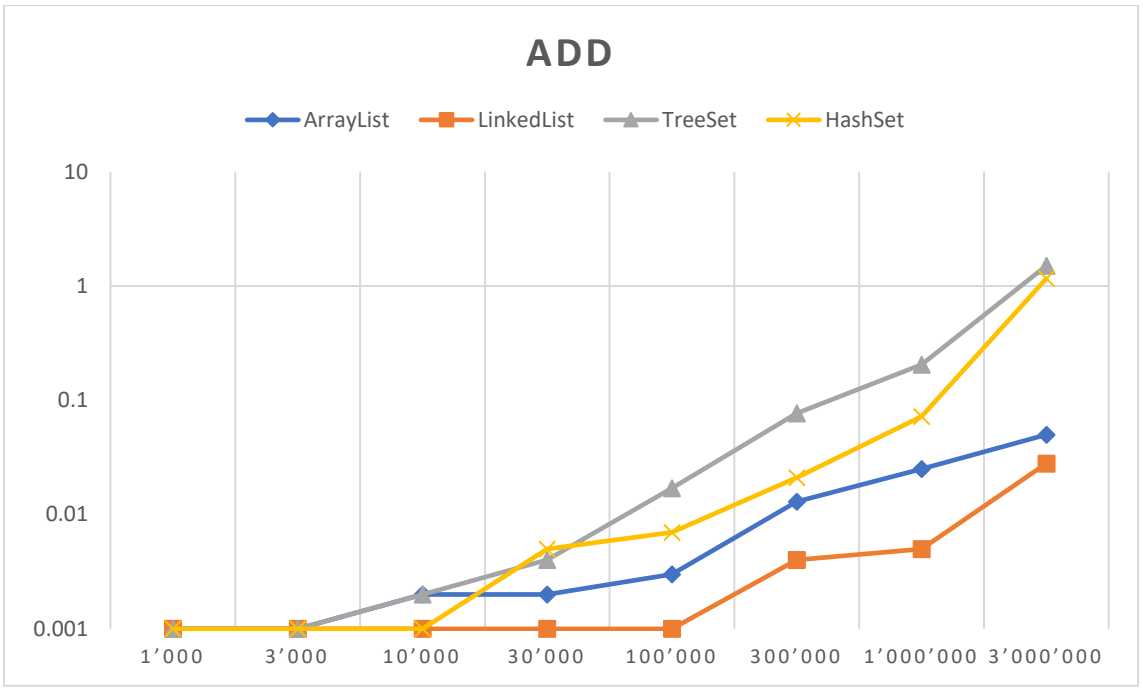
2.7 Search

Die Search Operation sucht ein zufällig ausgewähltes Element in der Datenstruktur.

3 Ergebnisse

Die Ergebnisse der Tests sind in den folgenden Grafiken dargestellt. Die Zeit zur Ausführung in Sekunden (X-Achse) wird in einer logarithmischen Skala ausgewiesen.

3.1 Zeit für Add



X-Achse: Zeit in Sekunden / Y-Achse: Datenmenge (Die Daten stammen aus eigener Messung in Java vom 29.11.2023)

3.1.1 Erklärung:

Die konstante Ladezeitentwicklung in der ArrayList ist durch die zugrunde liegende Datenstruktur, ein dynamisch wachsendes Array, bedingt.

Bei der LinkedList könnte die unregelmäßige Entwicklung auf sequenziellen Zugriff bei großen Datenmengen zurückzuführen sein.

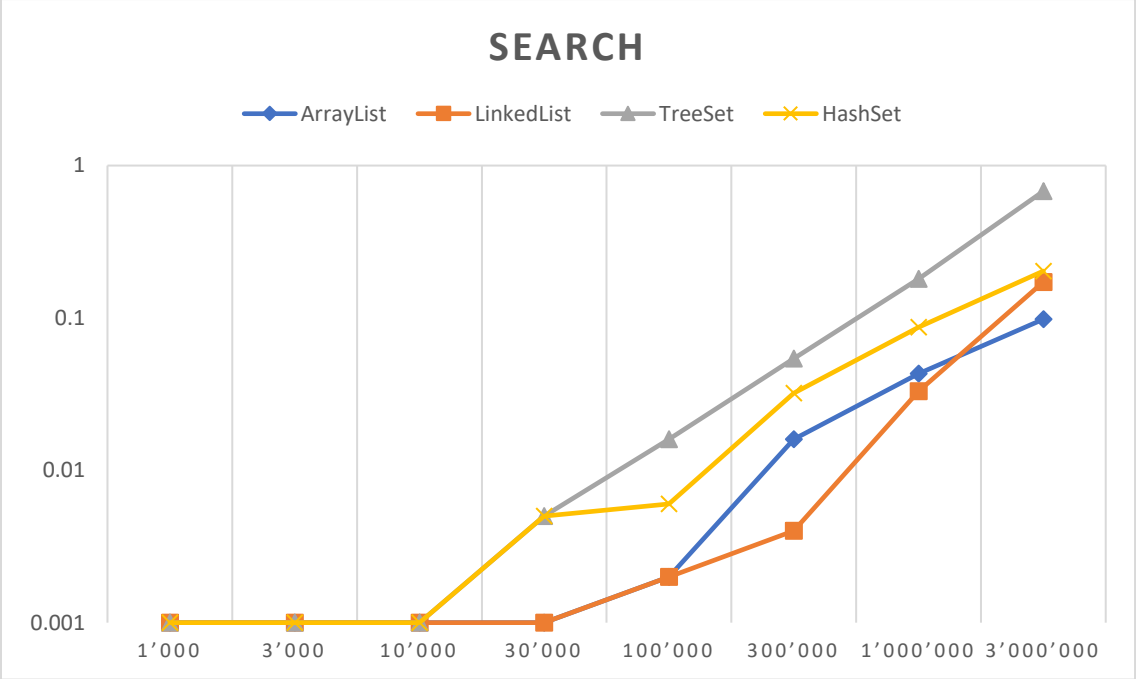
Das konstante Verhalten der Ladezeiten im TreeSet ist erwartet, da es auf einem balancierten Binärbaum basiert.

Im Gegensatz dazu weist das HashSet eine überproportional steigende Kurve auf, was jedoch im Kontext bestimmter Datenmengen innerhalb der erwarteten logarithmischen Zeitkomplexität liegt.

3.1.2 Überraschungen:

Die unregelmäßige Entwicklung der Ladezeiten in der LinkedList, besonders der Anstieg von 300'000 auf 1'000'000 Daten, überrascht.

3.2 Zeit für Search



X-Achse: Zeit in Sekunden / Y-Achse: Datenmenge (Die Daten stammen aus eigener Messung in Java vom 29.11.2023)

3.2.1 Erklärung:

Die ArrayList zeigt eine unterproportionale Zunahme der Ladezeiten mit steigender Datenmenge, wodurch die Kurve ab 300'000 Datensätzen abflacht. Dies ist auf eine effiziente Datenverwaltung in der ArrayList zurückzuführen.

Die LinkedList hat Schwierigkeiten bei der Suche in großen Datenmengen, was zu einem steilen Anstieg der Ladezeiten führt. Dennoch bleibt die LinkedList bis zu 1'000'000 Datensätzen die schnellste Methode zur Datenspeicherung.

Das TreeSet weist einen linearen Anstieg der Ladezeiten auf und ist die langsamste Methode zum Abrufen eines Datensatzes. Dies entspricht den Erwartungen aufgrund der balancierten Binärbaumstruktur.

Das HashSet zeigt einen flacheren Anstieg der Ladezeiten ab 300'000 Datensätzen und erreicht eine Konstanz ab 3'000'000 Datensätzen, ähnlich wie die ArrayList. Dies könnte auf eine effiziente Hashfunktion und Kollisionenbehandlung hinweisen.

3.2.2 Überraschungen:

Die unterproportionale Zunahme in der ArrayList und der flache Anstieg im HashSet ab 300'000 Datensätzen sind überraschende Beobachtungen, die auf spezifische Optimierungen oder Effizienzen in der Implementierung hinweisen könnten. Ebenso überrascht die Tatsache, dass die LinkedList trotz ihrer Schwierigkeiten bei der Suche bis zu 1'000'000 Datensätzen die schnellste Methode bleibt.

4 Fazit

4.1.1 "Add"-Operation:

Die Untersuchungen zeigen, dass die ArrayList eine konstante Ladezeitentwicklung aufweist, bedingt durch ihre effiziente Datenverwaltung. In der LinkedList und im HashSet sind unregelmäßige Entwicklungen zu beobachten, erklärbar durch sequenziellen Zugriff bzw. spezifische Herausforderungen bei der Hashfunktion. Das erwartete Verhalten im TreeSet bestätigt die logarithmische Zeitkomplexität aufgrund seiner balancierten Binärbaumstruktur.

4.1.2 "Search"-Operation:

Die ArrayList zeigt eine unterproportionale Zunahme der Ladezeiten, was auf effiziente Datenverwaltung hindeutet. Die LinkedList bleibt bis zu 1'000'000 Datensätzen trotz Suchschwierigkeiten die schnellste Methode. Das erwartete lineare Verhalten im TreeSet und der flache Anstieg im HashSet ab 300'000 Datensätzen entsprechen den erwarteten Eigenschaften dieser Datenstrukturen.

4.1.3 Insgesamt

Insgesamt verdeutlichen die Ergebnisse, dass die Wahl der geeigneten Datenstruktur stark von den spezifischen Anforderungen und der Art der Operationen abhängt. Überraschungen in den Beobachtungen weisen auf potenzielle Optimierungsmöglichkeiten oder Implementationseffekte hin, die weitere Untersuchungen rechtfertigen.