Programmieren in Java:
Visualisierung von Sortieralgorithmen

von
© *Kai Richard König (12.03)*

07 2011

FOS/BOS Technik München

Zusammenfassung

Dieses Dokument soll Aufschluss über die Überlegungen hinter der Entwicklung und Programmierung einer Java-Swing Desktop-Application zur Visualisierung von Sortieralgorithmen geben.

Inhaltsverzeichnis

Kapitel 1

Grobkonzept

1.1 Was macht das Programm ?

Das Java-Programm soll anhand von Visualisierungen die Funktionsweise von fünf[1] unterschiedlichen Sortieralgorithmen darstellen. Zur Verdeutlichung der Vorgänge wird jedem Algorithmus ein Balken-Diagramme zugewiesen, bei dem jeder Balken einem Zahlenwert entspricht, vergleicht der Algorithmus nun zwei Zahlen werden die korrespondierende Balken farblich hervorgehoben. Entscheidet der Algorithmus das einer der Balken vorschoben werden muss um alle Zahlenwerte in eine korrekte, aufsteigende Reihenfolge zu bringen so wird auch diese Operation im Balken-Diagramm durch einen farbliche Hervorhebung visualisiert.

Um die dargestellten Algorithmen noch besser verstehen zu können, laufen alle Operation synchronisiert und parallel ab, das heißt jeder Algorithmus kann pro Zeit nur eine Änderung an den Zahlenwerten durchführen. Diese Restriktion stellt sicher das die Ergebnisse reproduzierbar und vergleichbar bleiben. Dazu hat der Benutzer noch die Möglichkeit spezielle Zahlenwert/Balken Kombinationen auszuwählen um schwächen und stärken einzelner Algorithmen besser sichtbar zu machen. Denkbar wären zum Beispiel Kombinationen die in umgekehrter Reihenfolge vorliegen oder welche bei denen nur wenige Werte an der „falschen“ stelle sind oder auch solche bei denen es viele gleiche Werte gibt.

Wichtig ist vorallem das dem Anwender bei der Betrachtung klar wird welcher Algorithmus sich für welche Problemstellung am besten eignet. Erreicht wird das durch ein Ranking welches am Ende der Laufzeit jedes Algorithmus über das zugewiesen

[1] Bubble sort(Type: Exchange sort) — Quicksort(Type: Exchange sort) — Heapsort(Type: Selection sort) — Insertion sort(Type: Insertion sort) — Merge sort(Type: Merge sort) —

Balken-Diagramme gelegt wird. Auf der Fläche werden der Rang, sprich wie lange der Algorithmus im vergleich zu den anderen gebraucht hat, eine Diagramm bei dem der Grad der „Sortiertheit“ über die Zeit dargestellt wird, sowie die Anzahl durchgeführter Operationen zu finden sein.

Kapitel 2

Fachkonzept

2.1 Begriffs- und Notationsklärung

2.1.1 Big-O-Notation

$$O(n)$$

Diese Notation dient der Angabe von Effizienz von Algorithmen man könnte auch sagen das damit die maximale Anzahl Schritte im schlechtesten,durschnitts,besten Fall oder auch Worst/Average/Best-Case-Scenario angeben wird.
Also z.b.: $O(n^2)$ bedeute das der Algorithmus bei einem Array von 4 Integern maximal 16 Schritte benötigt bis das Array sortiert ist - oder eben weniger da $4^2 = 16$. Wir werden von dieser Notation im folgenenden ein paar mal gebrauch machen um die Algorithmen zu beschreiben. Mathematisch korrekter wäre aber $O(f(n))$ da dem n noch ein konstanter Faktor, den Algorithmus betreffend, vorsteht.

2.1.2 Die Applikation oder App

Mit der „Applikation“ oder auch „Applikation“ ist im folgenden die von mir, Kai Richard König, Java-Desktop-Application zur visualisierung von Sortieralgorithmen gemeint.

2.1.3 Algorithmus

> *" Ein Algorithmus ist eine aus endlich vielen Schritten bestehende eindeutige Handlungsvorschrift zur Lösung eines Problems oder einer Klasse von Problemen. "()*

2.2 Wozu Visualisierungen von Algorithmen ?

Ohne jetzt eine Beweis zu führen, lässt sich behaupten das der Mensch und sein Gehirn in der Lage sind Informationen in visueller Form wesentlich schneller zu absorbieren als in geschriebener Form. Darum macht es auch durchaus Sinn die Abläufe eines Sortieralgorithmus zu visualisieren anstatt den Vorgang in textueller Form oder gar in Programmcode zu beschreiben. Außerdem sind Algorithmen ein wichtiger Bestandteil von Informatik-Studiengängen und auch deshalb es ist nicht zu verachten wenn Information durch soviele Kanäle transportiert werden wie möglich - einer davon könnte eben auch die Vorgangvisualisierung sein. Um nun wieder auf den eigentlich Inhalt dieser Seminararbeit zurück zuschwenken, möchte ich in den folgenden Abschnitte auf die einzelen in der Applikation vorkommenden Algorithmen eingehen, sie vorstellen und vergleichen.

2.3 Verwendete Algorithmen

2.3.1 Bubblesort

Der Bubblesort Algorithmus ist der einfachste unter den Sortieralgorithmen und ist dementsprechend langsam. Ganz einfach weil der Algorithmus einen ziemlich navien Ansatz wählt. Jedes Element eines Arrays wird mit dem nachfolgenden verglichen, ist es größer so tauschen die beiden Element den Platz. Dieser Vorgang wird dann solange wiederholt bis keines der Element mehr den Platz wechselt. Diese Herrangehensweise impliziert quasi das die Prüfung ob das Array sortiert ist oder nicht schon eingebaut ist. Ebenso ergibt diese Handlungsvorschrift eine „avarage case performance“ von $O(n^2)$. Würde Bubblesort nun versuchen ein schon sortiertes Array zu sortieren so bräuchte der Algorithmus genau die Anzahl Schritte wie das Array groß wäre und daraus würde sich dann eine „best case performance“ von $O(n)$ ergeben. (vgl. H. W. Lang, *Sortierverfahren Bubblesort*)

2.3.2 Quicksort

Quicksort ist, wie der Name schon sagt, ein sehr schneller Algorithmus mit einer einer Average-Case-Performance von $O(n \cdot log(n))$. Die Funktionsweise und der Aufbau ist anders als gedacht eher simpel. Die gute Performance wird durch eine Verfahren erreicht das sich „Teile und herrsche“ (lat. Divide et impera!, engl. Divide and conquer) nennt, dabei wird das Problem in kleinere Probleme unterteilt, diese werden dann gelöst und am Ende wieder zusammengeführt. Konkret wählt der Algorithmus zu erst ein Orientierungselement aus, in den meisten fällen bietet sich dazu das element in der Mitte an. Nun werden alle Element die kleiner sind als das Orientierungselement nach (tatsächlich benutze ich gerad die app um zu verstehen was da eigentlich los ist ‘) links und jene die größer sind nach rechts verschoben. Anschließend wird der Vorgang mit den Unterprobleme, also der linken und der rechten Seite, solange wiederholt, sprich der Algorithmus wird rekursiv aufgerufen, bis die länge der auftretenden Unterprobleme zwei erreicht. In der Praxis erweist sich ein gut implentierter Quicksort-Algorithmus als das schnelltste Sortierverfahren. (vgl. H. W. Lang, *Sortierverfahren Quicksort*)

2.3.3 Heapsort

Der Heapsort hießt deswegen so weil er ein Problem welches zuerst in einen Heap überführt wird sortieren kann. Heap bedeutet im Englischen „der haufen“ oder auch „die Menge“ aber eigenlich ist damit ein Binärbaum (engl. Binary Tree, Binary Heap) gemeint. Ein Binärbaum ist ein Datenstruktur bei der es sogenannte Konten (engl. leafs,nodes) gibt die maximal zwei Unterknoten haben können. Diese Struktur bringt Eigenschaften mit sich die dieser Algorithmus für sich nutzt. Der Binary Heap wird von links nach recht und von oben nach unten gefüllt, das heißt erst wird einen sogenannte Wurzel (engl. root node) angelegt die Wurzel bekommt dann zwei Nodes und so weiter bis das Problem komplett im Baum präsent ist. Zugleich wird ein Regelwerk umgesetzt das besagt das ein Knoten keinen Unterknoten haben darf dessen Wert größer ist als der eigene. Folglich steht an der Wurzel des Baumes immer der größte Werte. Entnimmt mann diesen wird der Baum nach dem beschrieben Regelwerk umgeordnent und demgemäß steht wieder das größte Element an der Wurzel. Diesen Vorgang wird wiederholt bis der Binary Heap abgebaut ist. Im ganzen ergibt sich daraus eine „avarage case performance“ von $O(n \cdot log(n))$. (vgl. H. W. Lang, *Sortierverfahren Heapsort*)

2.3.4 Insertionsort

Beim Insertionsort (englisch insertion ‚Einfügen‘ und englisch sort ‚sortieren‘) handelt es sich um einen relativ simplen Algorithmus. Das Prozedere in welchem das Problem