Team: 04

Florian Kletz (#2093735), Micha Severin (#2093379)

AI-3 AD — Praktikum #3

Aufgabenaufteilung:

Beide Teammitglieder waren Maßgeblich an allen Abschnitten beteiligt. Im Besoderen, bei der Verifikation ob Methoden das gewünschte Ergebnis liefern und richtig arbeiten, sowie bei Lösungsansätzen und Ideen bei Fehlern. Grob gegliedert wurde es wie folgt:

Florian Kletz hat bearbeitet:

- Erweiterung von Zwei-Wege- zu Mehr-Wege-Mergesort

Micha Severin hat bearbeitet:

- Analyse (Zeit- u. Zugriffsmessung) des Algorithmus

Quellenangaben:

1. JavaDocs

Begründung für Codeübernahme

1. Es wurde kein Code aus anderen Quellen übernommen.

Bearbeitungszeitraum:

Florian Kletz	Montag 20.05.2013 09:00-18:00 Dienstag 21.05.2013 08:00-10:00 Freitag 24.05.2013 17:00-21:00
Micha Severin	Montag 20.05.2013 11:00-17:00 Donnerstag 23.05.2013 12:00-18:00 Samstag 25.05.2013 11:00-16:00
Gemeinsam	Samstag 25.05.2013 13:00-18:00 Sonntag 26.05.2013 12:00-16:00

Aktueller Stand:

- Mehrwege Algorithmus ist implementiert und getestet.
- Zeitmessung der Laufzeit des Algorithmus.
- Zugriffe auf die Ein- u. Ausgabebänder.

Probleme:

Das größte Problem war es unseren Zwei-Wege-Mergesort Algorithmus so umzuschreiben, dass dieser auch mit einer variablen Anzahl von Bändern (Mehr-Wege-Mergesort) arbeiten kann. Hierzu mussten große Teile des Codes neu geschrieben werden, was sehr zeitintensiv war.

Arbeitsweise des Algorithmus:

Für den Mehr-Wege-Mergesort Algorithmus werden Bänder benötigt. Es wird initial die Bandanzahl definiert. Bei zum Beispiel drei Bändern, werden drei Eingabe und drei Ausgabebänder erstellt.
Auf dem ersten Eingabeband muss die unsortierte Folge gespeichert sein. Dies können zum Beispiel Zahlen oder Buchstaben sein. Wir werden nur eine Implementierung mit Integer Zahlen ermöglichen.

Die Bänder werden mit einer Klasse erstellt. Die Länge der Folge kann das Fassungsvermögen des Arbeitsspeicher überschreiten, somit werden die Bänder aus das Dateisystem ausgelagert. Aufgrund des Fassungsvermögens des Arbeitsspeichers ist es nicht möglich eine interne Datenstruktur wie zum Beispiel ein Array zu verwenden.

Die zu sortierende Folge wird durch Zufallszahlen generiert und auf das erste Eingabeband gespeichert. Nun werden beim initialen Run sortierte Folgen mit der Runlänge auf die Ausgabebänder geschrieben. Diese sortierten Teilfolgen werden auf die zur Verfügung stehenden Ausgabebänder geschrieben. Nachdem die Ausgabebänder alle die selbe Länge haben, wird im nächsten Schritt wieder beim ersten Ausgabeband begonnen. Um die Teilfolgen zu sortieren würde sich der Bubblesort Algorithmus anbieten. Es wäre aber grundsätzlich möglich verschiedene Sortier-Algorithmen zu verwenden. Die Runlänge ist variabel.

In dem eigentlichen Mergesort-Algorithmus werden dann zwei dieser Teilfolgen zu einer größeren sortierten Teilfolge vereinigt. Deren Länge verdoppelt sich somit. Diese Teilfolgen werden auf den freien Ausgabebändern gespeichert. Nachdem die Eingabebänder abgearbeitet sind, findet ein Wechsel zwischen Eingabe und Ausgabebändern statt. Ein- u. Ausgabebänder werden im Wechsel gelesen und beschreiben. Nach jedem Wechsel werden die Bänder zurückgespult und es wird wieder bei dem Index 0 begonnen.

Die gesamte Sortierung ist dann abgeschlossen, wenn die Runlänge größer als die initiale Länge des ersten Eingabebandes ist oder ein Ausgabeband die gleiche Länge wie die initiale Länge hat.

Optimierungen des Algorithmus

Zwei-Wege-Mergesort → Mehr-Wege-Mergesort

Die wichtigste Optimierung ist die Erweiterung von Zwei-Wege-Mergesort zu einem Mehr-Wege-Mergesort. Hierfür werden wir Anzahl der Bänder dynamisch halten und die benötigten Wechsel zwischen den Bändern jeweils neu berechnen. Hierdurch lässt sich das Sortieren mittels mehrerer Parameter verändern, um so die effizienteste Konfiguration zu ermitteln. Diese Optimierung ist grundlegend für alle weiteren Optimierungen.

Verwendung von Buffer Reader/Writer

Eine Optimierung ist es, nicht wie bisher jeweils eine Zahl vom Eingabeband einzulesen die für die Operation benötigt wird, sondern direkt eine variable Anzahl dieser Einzulesen um so die Zugriffe auf die Bänder zu minimieren. Hierfür könne man einen Buffer Reader benutzen. Dieser liest mehrere Zahlen aufeinmal ein und verwaltet diese dann. Das selbe Prinzip lässt sich auf das Schreiben der Ausgabebänder mit einem Buffer Writer anwenden.

Erkennung von leeren Eingabebändern

Bisher werden die Methoden unabhängig ob auf den Bänden noch Zahlen stehen aufgerufen. Haben wir jedoch eine so große Menge von Eingabebändern, das nicht alle mit Zahlen beschrieben sind, müssen wir für diese die entsprechenden Methoden nichtmehr aufrufen (die dann keine Aktionen mehr ausführen). Dies würde eine Performancesteigerung zur Folge haben. Das Problem lässt sich lösen, wenn bemerkt wird das ein Eingabeband leer ist, denn dann können auf den folgenden Eingabebänder auch keine Zahlen mehr gespeichert sein.