AD-Aufgabe02-Gruppe-SawatzkiGlake 10.10.2013

Daniel Glake, Fabian Sawatzki

Praktikumsgruppe\_2

Übungsaufgabe 2.1.4

|  |  |
| --- | --- |
| Implementationsdetails zur Listenimplementation | Zur Realisierung der Matrix- Strukturen wurden uns drei Vorgaben gemacht, die allesamt umgesetzt wurden. Die zweite und die dritte Variante haben eine andere Form von Inhalt, den wir folgendermaßen entworfen haben: |
| II Iplementation (MatrixArrayList) | Realisierung, über ein Array, an dem Liste angefügt wurden, in den wiederrum nur Daten gespeichert werden, die ungleich 0 sind. Das Abspeicher der konkreten Position erfolgt über ein Tupel, dessen erste Komponente einen Werte, für die Spaltenposition hat. Dafür haben wir eine eigene Klasse Tupel entwickelt, die dies übernimmt. Eine Komponente für die Spaltenposition und ein Attribut, für den tatsächlichen Zahlenwert. Im Konstruktor, der zweiten Implementation ist es möglich direkt Werte mit zu übergeben. Diese Werte werden beim Eintrag in die Matrix dann abgefragt und im Falle von 0 nicht gespeichert. |
| III Implementation (MatrixList) | Die dritte Implementation soll über eine einzige Liste realisiert werden und stellt daher eine flachgeklopfte Matrix dar. Zur Untersuchung von Werten in der Matrix, muss diese im schlimmsten Fall, bei einer vordefinierten Größe von n x n, immer pn² durchlaufen. Das p steht hierbei für die Wahrscheinlichkeit von möglichen fehlenden 0 Werten. Eine Zeile in einer Matrix wird im besten Fall wohlmöglich leer sein oder auch nur ein Element haben, je nachdem wie p gewählt wird. Die Realisierung erfolgt einfach über eine Listenstruktur von Java. Ähnlich, wie in der zweiten Implementation werden auch hier nur Werte gespeichert, die ungleich 0 sind. Die Positionsangaben zu den Matrix- Elementen erfolgt über das Einfügen eines Tripels, dessen erste Komponente die Zeilenposition angibt und dessen zweite Komponente die Spaltenposition angibt. Der eigentliche Zahlenwert wird in der dritten Komponente abgelegt. |

Übungsaufgabe 2.1.7

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Aufgabenstellung: | Wie viel Platz wird mit den beiden Listenimplementationen II und III wirklich eingespart? | | | |
| Teilaufgabe a) Erzeugung von n x n Matrizen mit Vergleich des Platzbedarfs der Implementationen II und III über die Wahrscheinlichkeit p und einer stabilen Mittelwertbildung über die Durchläufe t: | Hypothese: Je Nachdem wie man die Aufgabenstellung auslegt, erwarten wird entweder die gleiche Anzahl an Elementen in den Listen, sofern jedes Tupel bzw. Tripel als ein Element zählt. Ansonsten wird die zweite Implementation der Matrix im Vergleich zur dritten Implementation einen um ein Drittel geringeren Platzbedarf besitzen. Die letzten vier Ziffern unserer tatsächlichen Matrikelnummer zu nehmen, war uns leider unmöglich, weil der Rechenaufwand dadurch zu hoch war. Außerdem konnten wir das t nicht groß genug wählen, um ganze 5 Stellen stabil zu halten, sondern nur 4 Stellen. Alles andere hätte den zeitlichen Rahmen und die Stromrechnung gesprengt. ☺  Bestimmung des Platzbedarfs für verschiedene Wahrscheinlichkeiten sowie Implementationen gemessen an der Anzahl der tatsächlichen Elemente (nicht der Anzahl der Tupel bzw. Tripel) – n beträgt 1028: | | | |
| Implementation | | 1% | 5% | 10% |
| Matrix Array List (zweite Implementation) | | 21138 | 105636 | 211368 |
| Matrix List (dritte Implementation) | | 31707 | 158454 | 317052 |
| Matrix Array (erste Implementation) | | 1056784 | 1056784 | 1056784 |
|  | Ergebnis: Da sich die Zeilen bei der zweiten Implementationsvariante aus der jeweiligen Liste im Array ergeben, muss diese nicht mitgespeichert werden. Daher ist der Platzbedarf bei der zweiten Implementation um ein Drittel geringer als bei der dritten Variante (Spaltenanzahl und das Element müssen natürlich auch gespeichert werden), obwohl bei beiden die Elemente mit einem Wert von Null überhaupt nicht gespeichert werden müssen. | | | |
| Teilaufgabe b) Graphische Darstellung des Platzbedarfs, über die Elemente, in den Tripeln und Tupeln: | Die Anzahl an Durchläufen t betrug 700. Für die Größe der Matrizen wurde die Dimension von 1028 x 1028 gewählt. | | | |
| Teilaufgabe c)  Bestimmung der mittleren Speichermenge, abhängig von p, für alle drei Implementationen: | Im Mittel können wir eine Menge von p\*n Elementen, in einer MatrixArrayList speichern, bei einer Matrixgröße von n x n; p für die Wahrscheinlichkeit der Elemente ungleich 0. Für die MatrixList gilt, dass eine Menge pn² gespeichert werden kann, ebenfalls bei einer Größe n x n. Das MatrixArray speichert immer n x n Elemente ab, wenn die Matrix die Ausmaße n hat. | | | |
| Teilaufgabe d) Bestimmung des Overheads, der Listenimplementationen: | Der Overhead wird im Falle des Platzbedarfes von allen Elementen gebildet, dessen Wert Null ist. Schließlich werden diese zwar an den Konstruktor, bzw. die setValue-Methode übermittelt, jedoch werden diese nicht gespeichert und zählen daher auch nicht zu den primären Nutzdaten. Folglich wird der Overhead durch (1-p) \* n beschrieben. | | | |

Übungsaufgabe 2.1.8

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Aufgabenstellung: | Wie viel Zeit nehmen unsere Listenimplementationen tatsächlich in Anspruch? Wie groß ist der Zeit-Overhead? | | | | | |
| Teilaufgabe a)  Messung des Zeitaufwands, für die Variationen von p und anhand der Anzahl der Dereferenzierungen. | Hypothese: Wir würden erwarten, dass der Zeitbedarf bei der zweiten Implementation geringer ist als bei der dritten. Schließlich wird wenn ein neuer Wert hinzugefügt wird bei der dritten Implementation über die komplette Liste iteriert. um zu überprüfen, ob es diesen Wert bereits gibt. Bei der zweiten Implementation wird immer nur über die Liste, der jeweiligen Zeile iteriert. Der Zugriff auf die Zeilen hat eine Laufzeit von O(1).  Erneut konnten wir das t nicht groß genug wählen, um ganze 5 Stellen stabil zu halten, sondern nur 4 Stellen. | | | | | |
| Beobachtung für n = 30 bei einer Matrixmultiplikation: | | | | | | |
| Implementation | | | 1% | 5% | 10% | |
| MatrixArrayList (zweite Implementation) | | | 7948 | 39773 | 77150 | |
| MatrixList (dritte Implementation) | | | 235359 | 1173586 | 2322546 | |
|  | | Ergebnis: Die übliche Laufzeit der Matrixmultiplikation beträgt n**³.** Dies bestätigen unsere Tests mit der ersten Implementation, denn dort haben wir für eine 30 x 30 Matrix einen Referenzenzähler von etwa 27000. Für die zweite Implementation kommt nun noch ein n hinzu, denn schließlich müssen wir über die einzelnen Listen iterieren, um an die gewünschten Werte zu kommen. Allerdings fallen die Elemente gleich Null wiederum heraus. Insgesamt kommen wir bei der zweiten Implementation also auf eine Laufzeit von p \* n^4. Dies bestätigen auch unsere Beobachtungen. Bei der dritten Implementation kommen wir schließlich sogar auf eine Laufzeit von p \* n^5, denn dort müssen wir nicht nur nach der jeweiligen Spalte, sondern auch nach der jeweiligen Zeile suchen. Die Abweichung nach oben erklärt sich durch die Erstellung der Matrix, die der Referenzenzähler bei und mitberechnung, welche eine Laufzeit von p \* n (für Implementation zwei und drei) besitzt. | | | |
| Teilaufgabe b) Graphische Darstellung des Zeitbedarfs für die Werte von p: | | Die Anzahl an Durchläufen t betrug 100. Für die Größe der Matrizen wurde die Dimension von 30 x 30 gewählt. | | | |
| Wiederholung des Zeitbedarfs zur Matrixmultiplikation mit einer Angabe zur Streuung (hier die korrigierte Varianz): | | Die Anzahl an Durchläufen t betrug 100. Für die Größe der Matrizen wurde die Dimension von 30 x 30 gewählt. | | | |
| Zustäzliche Quellen | | Für diese Aufgabenblatt wurden keine anderen Quellen herangezogen. | | | |