Datenstrukturen und Algorithmen
Übung 6, Frühling 2011

31. März 2011

Diese Übung muss zu Beginn der Übungsstunde bis spätestens um 16 Uhr 15 am 14. April abgegeben werden. Die Abgabe der DA Übungen erfolgt immer in schriftlicher Form auf Papier. Programme müssen zusammen mit der von ihnen erzeugten Ausgabe abgegeben werden. Drucken Sie wenn möglich platzsparend 2 Seiten auf eine A4-Seite aus. Falls Sie mehrere Blätter abgeben heften Sie diese bitte zusammen (Büroklammer, Bostitch, Mäppchen). Alle Teilnehmenden dieser Vorlesung müssen Ihre eigene Übung abgeben (einzeln oder in Zweiergruppen). Vergessen Sie nicht, Ihren Namen und Ihre Matrikelnummer auf Ihrer Abgabe zu vermerken.

Theoretische Aufgaben

1. Professor Marley behauptet, dass eine erhebliche Performanzsteigerung erzielt werden kann, wenn wir das Verkettungsschema so modifizieren, dass jede Liste in sortierter Ordnung gehalten wird. Wie beeinflusst die durch den Professor vorgeschlagene Änderung die Laufzeit für erfolgreiches Suchen, erfolgloses Suchen, Einfügen und Löschen? **1 Punkt**

2. Betrachten Sie eine Hashtabelle der Grösse $m = 1000$ und eine zugehörige Hashfunktion $h(k) = \lfloor m(kA \bmod 1) \rfloor$ mit $A = (\sqrt{5} - 1)/2$. Berechnen Sie die Plätze, auf die die Schlüssel $51, 52, 53, 54$ und 55 abgebildet werden. **1 Punkt**

3. Betrachten Sie das Einfügen der Schlüssel $8, 25, 12, 11, 34, 22, 16, 54, 38$ in eine Hashtabelle der Länge $m = 11$ durch offene Adressierung mit der Hilfshashfunktion $h'(k) = k \bmod m$. Illustrieren Sie das Ergebnis des Einfügens dieser Schlüssel mithilfe linearen Sondierens, quadratischen Sondierens mit $c_1 = 1$ und $c_2 = 3$ sowie durch doppeltes Hashing mit $h_2(k) = 1 + (k \bmod (m - 1))$. **1 Punkt**

4. Zeigen Sie, dass die mittlere Anzahl von Hashtabellenplätzen, die bei einer erfolgreichen Suche (mit gleicher Wahrscheinlichkeit für alle Schlüssel) inspiziert werden, bei Hashing mit linearem Sondieren nicht von der Reihenfolge abhängt, in der die Schlüssel in die anfangs leere Hashtabelle eingefügt worden sind. Gilt die entsprechende Aussage auch für quadratisches Sondieren? **1 Punkt**

5. Machen Sie einen Vorschlag, wie der Speicherplatz von Elementen innerhalb der Tabelle zugewiesen und freigegeben werden kann, wenn alle unbenutzten Plätze in einer

Freiliste zusammengefasst werden. Angenommen, auf einem Platz kann eine Markierung und entweder ein Element und ein Zeiger oder zwei Zeiger abgespeichert werden. Alle Operationen auf dem Wörterbuch und der Freiliste sollten in einer erwarteten Zeit von $O(n)$ laufen. Muss die Freiliste doppelt verkettet sein oder genügt eine einfach verkettete Liste? **1 Punkt**

Praktische Aufgaben

In dieser Übung werden Sie eine Variante von Hashing entwickeln, um räumliche Daten zu verwalten. Als Ausganslage dazu stellen wir auf ILIAS Code zur Verfügung, der eine einfache Partikelsimulation implementiert. Nehmen Sie sich ein paar Minuten Zeit, um ihn zu studieren. Der Code simuliert eine Menge von Partikeln, die sich frei im zwei-dimensionalen Raum bewegen, bis sie mit einem anderen Partikel oder der Umgebung kollidieren. Bei einer Kollision werden die Partikel gemäss einem einfachen Modell abgelenkt. Die Simulation erfolgt über diskrete Zeitschritte. In jedem Schritt werden die Partikel gemäss ihrer Geschwindigkeit weiterbewegt, und bei Kollisionen wird zusätzlich die neue Geschwindigkeit und Richtung berechnet.

Eine naive Implementation der Kollisionsdetektion überprüft jedes Paar von Partikeln auf eine mögliche Kollision. Der Aufwand für die Kollisionsdetektion ist somit also $O(n^2)$, wobei n die Anzahl Partikel ist. Bei einer grösseren Anzahl Partikel wird dadurch der Aufwand zur Berechnung jedes Schritts in der Simulation schnell von der Kollisionsdetektion dominiert.

Ihre Aufgabe ist es, einen effizienteren Algorithmus zur Kollisionsdetektion zu entwickeln. Die Idee ist es, eine Datenstruktur zu verwenden, welche die Partikel gemäss ihrer räumlichen Position verwaltet. Weil die Partikel zufällig im Raum verteilt sind, kann die Position eines Partikels verwendet werden, um einen Hashwert zu berechnen ähnlich wie zu Beginn von Kapitel 11.3 im Buch beschrieben. Gegeben die x Koordinate des Partikels im Bereich $0 \leq x < w$ ist der Hashwert $h(x) = \lfloor xm/w \rfloor$, wobei m die Grösse der Hashtabelle ist. Ebenso kann für die y Koordinate ein Hashwert berechnet werden. Die beiden Hashwerte können nun als Indizes in eine zweidimensionale Hashtabelle verwendet werden.

Die zweidimensionale Hashtabelle entspricht einem (virtuellen) zweidimensionalen Gitter: jeder Partikel wird entsprechend seiner Position einer Gitterzelle zugeordnet. Damit kann die Kollisionsdetektion effizienter gemacht werden, indem jeder Partikel nur auf Kollisionen in seiner eigenen Gitterzelle und den unmittelbaren Nachbarzellen untersucht wird. Die Nachbarzellen müssen getestet werden, weil Partikel eine gewisse Ausdehnung haben und mit den Nachbarzellen überlappen können. Beachten Sie auch, dass die Partikel nach jedem Simulationsschritt entsprechend ihrer neuen Position aus der Hashtabelle entfernt und in die richtige Zelle wieder eingetragen werden müssen .

1. Modifizieren Sie die Klasse *BouncingBallsSimulation*, um den skizzierten Algorithmus zu implementieren. Verwenden Sie ein zweidimensionales Feld von verketteten Listen als Hashtabelle. Für die Listen können Sie die Java *LinkedList* Klasse verwenden. Ihre Hashtabelle ist dann vom Typ *LinkedList<Ball>[][]*. Verwenden Sie einen Iterator vom Typ *ListIterator<Ball>* um die Listen zu traversieren. Dieser Iterator erlaubt Ihnen mit seiner Methode *remove()* effizient das letzte besuchte Element aus der Liste zu entfernen.

Drucken Sie nur Ihre modifizierte Klasse *BouncingBallsSimulation* aus und geben Sie sie ab. **3 Punkte**

2. Testen Sie Ihren Algorithmus für verschiedene Grössen der Hashtabelle und verschiedene Anzahl Partikel. Stellen Sie die Zeitmessungen in einer Tabelle zusammen. Was ist jeweils die optimale Grösse der Hashtabelle? Was könnte der Grund sein, dass die Geschwindigkeit bei grösseren Tabellen wieder abnimmt? **2 Punkte**