Protokoll „Hashtabelle“

by Selina Brinnich und Alexander Dietrich

Aufwandsabschätzung:

Datenstruktur	Array	Lineare Liste	Hashtabelle
Suchaufwand	O(N)	O(N)	O(1)
Aufwand Einfügen	O(N)	O(N)	O(1)
Löschen	O(N)	O(1)	O(1)

Hashfunktion (hashtable.cpp -> Zeile 13-37):

Die Hashfunktion die wir in unserem Programm verwendet haben, wird in 3 Teilen abgearbeitet die folgend nun näher erläutert werden:

1. Ein String mit beliebiger Länge wird an die Funktion übergeben, der String wird in Teile bestehend aus 4 Zeichen zerlegt um die String-Teile separat behandeln zu können.
2. Jeder aus 4 Zeichen bestendender String wird erneut abgespeichert und Buchstabe für Buchstabe mit dem Wert 256 (8 Bit) multipliziert.
3. Wenn der String vollständig durch multipliziert wurde, wird der Wert modulo der Arraygröße zurückgegeben. Dieser Wert repräsentiert den Platz in der Hashtabelle.

Sollte der String nicht durch 4 teilbar sein, wird der letzte Teilstring extra mit der verfügbaren Länge mittels gleichem Berechnungsschema berechnet.

Kollisionsbehandlung (hashtable.cpp Zeile 50-62):

Falls eine Kollision unter Einträgen verursacht wird, verwenden wir eine alternierende quadratische Sondierung, um einen geeigneten Platz zu finden. Somit ist der Aufwand, der im schlimmsten Fall entsteht, niemals höher als O(N), da nach einem Durchlauf bereits alle Hashtabellenpositionen durchsucht wurden.

Kursdaten (Stock.h):

Die Kursdaten verwalten wir mit 2 Klassen, der Pricedata- und der Stockklasse. In der Klasse Stock befinden sich Kürzel, Name, und ein Array der Klasse Pricedata um maximal 30 Tage der Aktienentwicklung zu speichern. Die Klasse Pricedata besteht aus sämtlichen Informationen die eine Aktie enthält (Datum, Eröffnungswert, Höchstwert, …). Jede dieser Aktien hat wiederum einen Platz in unserem Hashtabellen Array, das 1999 (Primzahl) Plätze zur Verfügung hat. Somit können wir von maximal 1999 verschiedenen Aktien Daten der letzten 30 Tage abspeichern und graphisch darstellen. Um möglichst effizient zu arbeiten, sollten jedoch nicht mehr als 1000 Einträge abgespeichert werden.

Löschalgorithmus (hashtable.cpp -> Zeile 72-96):

Verwendet wird beim Löschen von Elementen der gleiche Algorithmus wie beim Suchen beziehungsweise bei Kollisionen. Wir gehen mit dem Ausgangspunkt vom gehashten Wert der Eingabe mittels alternierender quadratischer Sondierung durch unsere Hashtabelle, bis wir den richtigen Eintrag gefunden haben. Falls der Eintrag nicht gefunden wurde, dann wird nach einem maximalen Aufwand von O(N) der Suchlauf beendet, da die gesuchte Aktie nicht in der Hashtabelle verfügbar ist.