# Protokoll Hashtabelle – Aktienprogramm

### Aufgabenstellung:

Implementieren Sie ein Programm zur Verwaltung von Aktienkursen.Um schnell nach einer bestimmten Aktie suchen zu können, sollen die Aktien in einer Hashtabelle verwaltet werden.Pro Aktie sollen Name, Wertpapierkennnummer (WKNals String)und Kürzel(auch ein beliebiger String)gespeichert werden. Weiters sind pro Aktie die Kursdaten der vergangenen 30 Tage zu verwalten mit folgenden Informationen:(Date,Open,High,Low,Close,Volume,Adj Close)

Das Programm wurde in C++ realisiert. Die von uns erzeugte Klasse Hashtable beinhaltet folgende Funktionen:

* Find\_next\_prime
* Create\_stock
* Hash\_function
* Quadratic\_probing
* Quadratic\_probing\_tag
* Quadratic\_probing\_name
* Find\_stock\_tag
* Find\_stock\_name
* Error
* To\_upper
* Add
* Del
* Import\_data
* Search
* Plot
* Save
* Load
* Quit

Bei Programmstart wird eine Hashtabelle mit zwei Arrays erstellt: eines dient der Namensuche, während das andere die tatsächlichen Werte der Aktien speichert. Sie fungieren als Fundament zur Suche, zur Löschung und zur Verwaltung der Daten, die gespeichert wurden.

Die Funktion Find\_next\_prime erzeugt dabei die nächstgrößere Primzahl für den festgelegten Wert im Programm und setzt sie als Limit für die Hashtabelle und der Modulo-Funktion an. Zur Kollisionsvermeidung soll quadratische Sondierung angewendet werden.

### Hashfunktion

Folgende Berechnungsfunktion wurde gewählt, um die Werte gleichmäßig in die Tabelle einzufügen:

y ist eine Primzahl und s der ASCII-Wert der Buchstaben. Es wurde weiters überlegt, der Wert des Divisors erhöht werden soll. Aufgrund der niedrigen Anzahl an maximalen Aktien erschien dies redundant.

Im Kollisionsfall wird zu H(s) ein quadratisch wachsender Wert addiert, um eine möglichst große Verteilung der Werte in der Hashtabelle zu erzielen.

### Kollisionserkennung

Die Kollisionserkennung erfolgt innerhalb von zwei Arrays. Zur Prüfung einer Kollision im Datenarray

Für die Kollisionserkennung wird an der Stelle, wo der Wert einzufügen ist überprüft, ob ein Wert bereits vorhanden ist. Mithilfe einer bool‘schen Member-Variable wird überprüft, ob der Wert bereits zuvor gesetzt wurde oder nicht.

Sollte diese auf wahr gesetzt sein, wird überprüft, ob es sich um die gewünschte Position handelt oder nicht. Falls nicht, wird solange weiter sondiert, bis man zur gewünschten Stelle landet. Dies kann eine freie Position sein oder ein gesuchter Wert.

### Verwaltung der Kursdaten

Die Verwaltung der Kursdaten wird über eine eigene Datenstruktur umgesetzt. Dafür wurde eine eigene Struktur und Klasse erzeugt, die das Importieren einer CSV Datei managed und diese Werte in einem dynamischen Array speichert. Die maximale Anzahl an Datensätzen beträgt dabei 30. Sollte dieser Wert überschritten werden, wird an der letzten Stelle ein weiteres Element hinzugefügt und das erste Element entfernt. Die verwendete Vector-Klasse in C++ macht dies ineffizient. Beim Hinzufügen eines Elementes wird zunächst ein Speicherbereich, der um den Anteil größer ist, um das neue Element zu speichern, allokiert. Anschließend werden alle vorhandenen Werte in den neuen Speicherbereich kopiert und der alte Speicherplatz freigegeben.

Bei der Erstellung eines Datenknotens erfolgt eine Aufforderung nach einem Aktiennamen, einem Aktienkürzel und einer eindeutigen Aktiennummer. Mithilfe dieser Daten wird in den jeweiligen Arrays überprüft, ob diese Daten bereits vorhanden sind. Im Falle eines bereits vorhandenen Datensatzes wird der Vorgang abgebrochen und gelangt zurück zum Hauptmenü. Andernfalls werden die Datenstrukturen an den zugehörigen Stellen befüllt. Die Stellen werden mithilfe der Hashfunktion und der quadratischen Sondierung im Kollisionsfall berechnet.

### Löschalgorithmus

Löschung der Daten folgt einem ähnlichen Prinzip der Erstellung der Daten. Zunächst wird mithilfe der find\_stock\_name Funktion eine Aktie mit demselben Namen gesucht. Existiert eine Aktie mit diesem Namen, wird ein Pointer in eine Membervariable der Klasse gespeichert. Aus dem Input muss nun der korrespondierende Eintrag in der Datenstruktur für die Namenssuche gefunden und gelöscht werden. Hierfür findet die Hash-Funktion ihre Verwendung, um die Adresse des Eintrags zu berechnen. Sollte an dieser Stelle jedoch ein anderer Wert gespeichert werden, wird der Wert über die quadratische Sondierung weiter gesucht. Die Datensätze werden anschließend in dem Aktien Array gelöscht bzw. in dem Namensarray entleert und zur Auffüllung markiert. Bei der neuen Erstellung eines Datenpunktes werden diese Lücken aufgefüllt.

### Aufwandsabschätzung

Hashtabellen haben den Vorteil, dass sie für die Suche, dem Einfügen und dem Löschen eine durchschnittliche Laufzeit von O(1) haben. Dies wird durch die Berechnung mithilfe der Hashfunktion ermöglicht. Dieser Wert ist nun der Speicherort des entsprechenden Datensatzes. Somit können Daten leicht eingefügt, gesucht und gelöscht werden. Im Worstcase ist die Laufzeit aber O(n). Dieser Wert kommt von der Notwendigkeit einer Sondierung, wenn sich zwei Hashwerte unterschiedlicher Datensätze gleichen. Bei einem Array ist die durchschnittliche Laufzeit für suchen, einfügen und löschen O(n) und der Worstcase ist ebenfalls O(n). Dabei ist es irrelevant ob in einem Array etwas eingefügt oder gelöscht werden soll da jedenfalls der Datensatz oder ein leeres Feld gesucht werden müssen. Im fall einer verketten liste, ist es beim davon anhängig ob der Datensatz am Anfang der Liste mit O(1) eingefügt werden soll oder mit O(n) am ende der Liste. Soll der Datensatz nämlich am Ende der Liste eingefügt werden muss nämlich die ganze liste traversiert werden. Die suche hingegen ist sowohl im durchschnittlichen als auch im Worstcase O(n). Dies kommt von der Notwendigkeit im Worstcase die gesamte Liste zu durchlaufen. Aus der Laufzeit der suche resultiert daher auch die des Löschens mit O(n), weil jedes zu löschende Element erstmal gesucht werden muss.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Einfügen | | Suchen | | Löschen | |
|  | Durchschnitt | Worstcase | Durchschnitt | Worstcase | Durchschnitt | Worstcase |
| Hashtabelle | O(1) | O(n) | O(1) | O(n) | O(1) | O(n) |
| Array | O(n) | O(n) | O(n) | O(n) | O(n) | O(n) |
| Verkettete Liste | O(1) oder O(n) | O(n) | O(n) | O(n) | O(n) | O(n) |

Zusammenfassend ist im durchschnittlichen Fall die Hashtabelle immer O(1), welches nur für die verkette Liste mit dem einfügen am Kopf der liste möglich ist. Für alle anderen Fälle ist die Laufzeit O(n).

Bei zunehmenden Füllgrad der Hashtabelle werden Kollisionen beim Einfügen häufiger wodurch es zu mehr Sondierungen kommt was die Laufzeit erhöht. Bei der quadratischen Sondierung ist zu beachten, dass die Tabelle, die doppelt so groß sein sollte als Datensätze gespeichert werden sollen. Damit soll garantiert werden, dass die Sondierung immer einen alternativ Platz für die Patensätze findet.