# Übung 1 – HashTabelle

**Protokoll**

Für das Projekt haben wir Python verwendet.

Die Hashtabelle verwendet folgende Hashfunktion:  
Diese ist die Java Hashfunktion, wir bekommen jedoch andere Ergebnisse als in Java (kein Integer-Overflow).

]

Der Hashwert wurde dann den Modulo der Länge unserer Tabelle dividiert um den Index der Tabelle zu berechnen.   
  
Als Länge unserer Tabelle haben wir uns für 2003 entschieden, da 1000 Aktien gespeichert werden sollten und 2003 die erste Primzahl der doppelten Größe ist. Die Tabelle soll max. 50% Füllgrad haben.

Im Falle einer Kollision wurde quadratische Sondierung implementiert. Wenn der Index, den wir von der Hashfunktion bekommen schon besetzt ist, so wurde immer der nächste freie Index gesucht. (index + i^2)  
  
Wir verwenden zwei Tabellen und speichern die Daten in beiden ab. Einmal wird als Schlüssel der Aktienname und einmal das Aktienkürzel verwendet.  
Durch die zwei Tabellen kann sowohl nach dem Name als auch nach dem Kürzel gesucht werden. Bei Löschen (Delete) werden beide Einträge gelöscht.

Beim Suchen wird wieder mit der Hashfunktion versucht den Index von dem übergebenen Schlüssel zu berechnen. Falls der Schlüsselname (oder -kürzel) nicht mit dem Eintrag im Index nicht übereinstimmt wird weiter gesucht (mittels quadratischer Sondierung).

In CSV-Dateien gespeicherte Aktien-Kursdaten können zu bestehenden Aktien geladen werden. Diese werden im data Attribut in der Klasse Stock (=Aktie) gespeichert und können als Diagramm geplottet werden. Zusätzlich sieht man den aktuellsten Kurseintrag bei der Suche nach einer Aktie.

Mit Save und Load werden alle Dateien im JSON-Format gespeichert bzw. geladen.  
  
Der Vorteil zu einer verketten Liste oder einem Array ist, dass in der Regel der Datensatz mit Hilfe des Schlüssels und der Hashfunktion direkt gefunden werden kann O(1) und nur im worst-case durch die ganze Tabelle gesucht werden muss O(n).

**Aufwandabschätzung:**

1. **\_\_init\_\_() :**   
   Initialisiert 2 Tabellen(symbol\_table) und (name\_table) mit Länge n.  
   O(n)
2. **add\_stock():**
   1. Objekt von Klasse Stock erstellen -> O(1)
   2. Hashcode berechnen -> O(n)
   3. Stock bei Index speichern -> O(1)
3. **find\_index():**  
   Worst-case : O(n)

Normal: O(1)

1. **search\_index():**  
   Worst-case : O(n)

Normal: O(1)

1. **hash\_code():**  
   Java hashCode()function  
   n = Länge von Key  
   O(n)
2. **delete\_stock():**Worst -case: O(n)   
   Normal: O(1)
3. **import\_stock()**:  
   m = Anzahl von Daten im file  
   O(m)

Quellen:

ALGOS Kursunterlagen

https://www.geeksforgeeks.org/quadratic-probing-in-hashing/  
  
https://www.hackerearth.com/practice/data-structures/hash-tables/basics-of-hash-tables/tutorial/  
  
https://www.freecodecamp.org/news/big-o-notation-why-it-matters-and-why-it-doesnt-1674cfa8a23c/