Team: 03, Sebastian Diedrich – Murat Korkmaz

Aufgabenaufteilung:

1. Aufgaben, für die Teammitglied 1 verantwortlich ist:

   (1) Skizze

   Dateien, die komplett/zum Teil von Teammitglied 1 implementiert/bearbeitet wurden:

   (1) Operationen: create, insert

   (2) Tests: Grenzfälle

2. Aufgaben, für die Teammitglied 2 verantwortlich ist:

   (1) Skizze

   Dateien, die komplett/zum Teil von Teammitglied 2 implementiert/bearbeitet wurden:

   (1) Operationen: create, insert

   (2) Tests: Grenzfälle

Quellenangaben: Vorlesung am 16.12.15, Folien AD_Hash.pdf (Prof. Klauck)

Bearbeitungszeitraum: 28.12.15 (6h), 06.01.16 (7h)

Aktueller Stand:

1. Skizze Version 1 (Rückmeldung durch Prof. offen)

FRAGEN: (gelb im Text weiter unten markiert)

a) In der zweiten Hashfunktion h‘(k) ist uns nicht genau klar, wie das k in der Klammer behandelt werden soll, da es sich ja um einen String handelt und keine Zahl
h‘(k) = 1+(k mod m‘)
Wir haben derzeit in unserer Skizze daher: h‘(k) = 1+(h(k) mod m‘) benutzt

b) Unter dem 6. Punkt steht: *das Suchen (find) und das Einfügen (insert) für Hashing nach Brent*
Soll auch das Suchen (find) nach Brent erfolgen? Dieses hat unserer Meinung nach keine Vorteile, da eventuell mehr Anfragen an die ADT-Struktur gestellt werden muss.

Skizze: (ab Seite 2)

Skizze Aufgabe 4:

Aufgabe: 4.1

Ziel: ADT-Hashmap implementieren

Quelle: http://pub.informatik.haw-hamburg.de/home/pub/prof/klauck_christoph/AD/AD_Hash.pdf

Folgendes soll immer gelten (Invarianten):

I) funktional
- Das Programm wird mit folgenden Parametern gestartet:
  strategy: enum (L -> lineares Sondieren, Q -> quadratisches Sondieren und
  B -> Double-Hashing nach Brent)
  filename: absoluter Pfad zur Textdatei (inklusive Filename)
- Als Dateien werden Text-Dateien im ASCII Format akzeptiert
- Das Programm soll die Anzahl der Vorkommen aller Wörter in der Textdatei zählen und in einer log-Datei ausgeben.

II) technisch
- In der Hashmap werden die Wörter gespeichert und ihre Anzahl
- Die einzelnen Wörter bilden die Hashkeys, im Value steht die Anzahl

Objektmengen:
- word: Wort als Key in der Hashmap
  (Sollten Umlaute und Sonderzeichen im Wort vorkommen, wird das Wort ignoriert)
- count: Anzahl der Vorkommnisse eines Wortes
- size: Anzahl der maximalen Speicheradressen in der Hashmap
  (Bedingung: natürliche Zahl ohne 0)
- hashmap: ADT-Hashmap (Key: word, Value: count)
- strategy: Auswahl des Sondierungsverfahrens

Folgende Operationen sollen bereitgestellt werden (semantische Signatur):

- *create*: eine leere Hashmap (mit size und strategy) erstellen
  (size x strategy -> hashmap)
  Fehlerbehandlung: Exception wird geworfen und es wird keine Hashmap erzeugt

- *insert*: Fügt ein Wort in die Hashmap ein oder erhöht die Anzahl, falls Wort bereits vorhanden um 1
  (hashmap x word -> hashmap)
  Fehlerbehandlung: ignorieren (Umlaute und Sonderzeichen im Wort vorhanden)

- *find*: Gibt das Value (Anzahl) eines vorgegebenen Keys (Wort) zurück
  (hashmap x word -> count)
  Fehlerbehandlung: 0 zurückgeben

Syntaxtische Vorgaben:

Dateiname: adtHashmap.jar

Klassenname: ADTHashmap

Anwendung der oben genannten Operationen:

create: ADTHashmap.create(size, strategy)

insert: <Objektname>.insert(word)

find: <Objektname>.find(word)

Kollisionsbehandlung in kurzer Zusammenfassung:

Verkettung (geschlossenes Hashing):
Trifft ein Schlüssel auf eine bereits belegte Adresse, wird dieser Schlüssel ebenfalls an diesem Speicheradresse abgelegt. Um dieses zu ermöglichen wird eine **Liste** an diesen Speicheradresse angelegt, in der sich unsortiert alle Schlüssel befinden. Wird nach einem Schlüssel gesucht, muss über die Liste iteriert werden.

Offene Adressierung:
Jeder Schlüssel erhält einen **eigene** Speicheradresse. Sollte beim Einfügen ein Schlüssel auf eine bereits belegte Speicheradresse treffen, wird mittels Sondierungsfunktionen (s.u.) nach einer freien Speicheradresse gesucht.

Hashfunktion:

Beschreibung:
Mittels Hashfunktion wird aus einem Schlüssel eine Hashadresse berechnet. Die Hashadresse gibt den Index in einem Feld an, auch Hashtabelle genannt.

Methode:
Es soll die Division-Rest-Methode als Hashfunktion für Java-Strings verwendet werden. Um Integer Überläufe zu vermeiden (max. 2^31-1) soll das Horner-Schema eingesetzt werden.

Pseudocode:
i natürliche Zahl, h=0; s Zeichenkette / Feld

1. for i = 0 to i < länge_von(s)
2. h = (h * 128 + s[i]) mod m;

Ergebnis: h

Quelle: http://pub.informatik.haw-hamburg.de/home/pub/prof/klauck_christoph/AD/AD_Hash.pdf

Beispiel:
s = hallo; m = 101

s[0] : h = (0 * 128 + 104) mod 101 = 3 // h
s[1] : h = (3 * 128 + 97) mod 101 = 77 // a
s[2] : h = (77 * 128 + 108) mod 101 = 66 // l
s[3] : h = (66 * 128 + 108) mod 101 = 72 // l
s[4] : h = (72 * 128 + 111) mod 101 = 35 // o

Ergebnis: Speicheradresse des Java-Strings „hallo“ ist 35.

Beschreibung der Sondierungsverfahren und deren Durchführung:

Es soll bei der Implementation der Operationen (find und insert) der ADT-Hashmap die offene Adressierung verwendet werden.

Definitionen:

- s : Sondierungsfunktion
- j : Anzahl der Versuche, eine freie Speicheradresse zu finden
- k : Schlüssel
- h : Hashfunktion
- h‘ : zweite Hashfunktion

Linear:

$h_j(k) = (h(k) + s(j,k)) \bmod m$ mit **$s(j,k) = j$**

Beispiel:

Sollte das Wort „hallo“ erneut als Schlüssel eingetragen werden, erfolgt dieses um eine Speicheradresse versetzt nach rechts.

$h_0(k) = (35 + 0) \bmod 101 = 35$
$h_1(k) = (35 + 1) \bmod 101 = 36$

Quadratisch:

$h_j(k) = (h(k) + s(j,k)) \bmod m$ mit **$s(j,k) = (-1)^j * j^2$**

Beispiel:

Sollte das Wort „hallo“ mehrmals als Schlüssel eingetragen werden, wird mittels der obigen quadratischen Sondierungsfunktion versucht eine freie Speicheradresse zu finden.

$h_0(k) = (35 + 0) \bmod 101 = 35$
$h_1(k) = (35 + (-1)) \bmod 101 = 34$

$h_0(k) = (35 + 0) \bmod 101 = 35$
$h_1(k) = (35 + (-1)) \bmod 101 = 34$
$h_2(k) = (35 + 4) \bmod 101 = 39$

Double-hashing mittels Brent-Verfahren (s.u.):

$h_j(k) = (h(k) + s(j,k)) \bmod m$ mit **$s(j,k) = j * h'(k)$**
$h'(k) = 1+(h(k) \bmod m')$
$m' = m - 2$

Beispiel:

Sollte das Wort „hallo“ mehrmals als Schlüssel eingetragen werden, wird mittels der obigen Sondierungsfunktion versucht eine freie Speicheradresse zu finden.

$h_0(k) = (35 + 0) \bmod 101 = 35$ // belegt
$h_1(k) = (35 + 36) \bmod 101 = 71$ // frei

$h_0(k) = (35 + 0) \bmod 101 = 35$ // belegt
$h_1(k) = (35 + 36) \bmod 101 = 71$ // belegt
$h_2(k) = (35 + 72) \bmod 101 = 6$ // frei

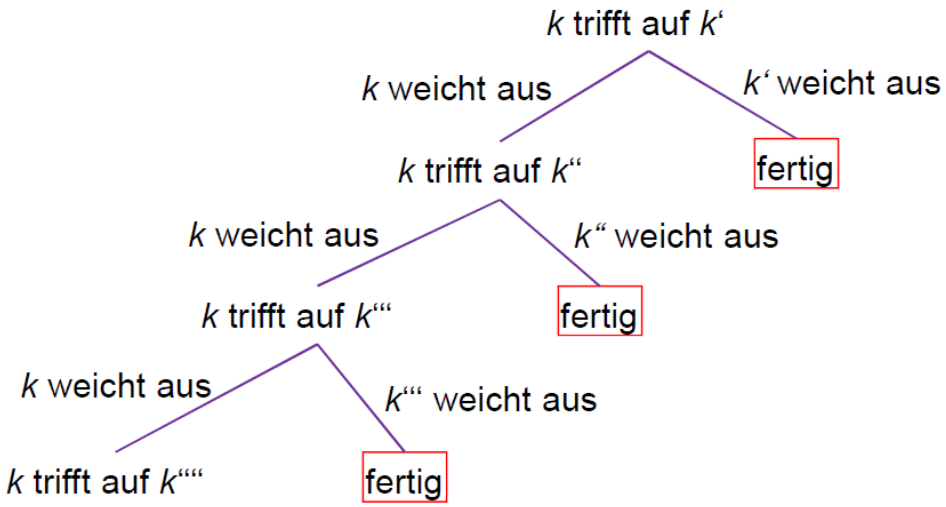
Verbesserung der erfolgreichen Speicheradressen-Suche (Brent-Verfahren)

Um beim Einfügen von Schlüsseln schneller eine freie Speicheradresse zu finden, kann eine Breiten- oder Tiefensuche (Verfahren nach Brent) benutzt werden.
Bei der Implementation der insert Methode soll das Verfahren nach Brent angewendet werden. Dabei muss jeder Schlüssel „wissen“ wie viele bisherige Versuche (j) er gebraucht hat, eine freie Speicheradresse zu finden. Trifft ein neuer Schlüssel auf eine belegte Speicheradresse, versucht zunächst der alte Schlüssel auf eine freie Speicheradresse auszuweichen. Dafür wird die Sondierungsfunktion mit j = j+1 aufgerufen.
Sollte dieses nicht zu einer freien Speicheradresse führen, weicht der neue Schlüssel mittels der Sondierungsfunktion mit j = j+1 aus. Trifft der neue Schlüssel auf eine freie Speicheradresse, wird diese benutzt. Falls die Speicheradresse erneut belegt ist, läuft der oben beschrieben Ablauf von vorne ab.

M1 : Pfad vom neuen Schlüssel

Brent’s Verfahren: verfolge nur (M1)

Quelle: http://pub.informatik.haw-hamburg.de/home/pub/prof/klauck_christoph/AD/AD_Hash.pdf



Test

Für jede Sondierungsstrategie sollen folgende Tests durchgeführt werden

1. J-Unit Tests
    - Diese sollen austauschbar sein
        i. Name der Jar: hashmJUt.jar
    - Input:
        i. Text mit 100 Wörtern:
        http://users.informatik.haw-hamburg.de/~klauck/AlguDat/texta.txt
        ii. Text mit 9790 Wörtern:
        http://users.informatik.haw-hamburg.de/~klauck/AlguDat/textb.txt
        iii. Text mit 0 Wörtern:
        selbstgenerierter Text
        iv. Text mit 100.000 Wörtern:
        selbstgenerierter Text
    - Output:
        i. ADT-Hashmap mit Anzahl unterschiedlicher Wörter (Key) und deren vorkommen im Text (Value)

Messung von Laufzeit

1. Implementierung einer Version der ADT um die Laufzeit der Operationen von insert und find durchführen zu können

2. Vorgaben für die Messungen: Anzahl der Wörter:
   Um die find-Operation vergleichbar zu machen, muss in jedem Text das Wort „dolore“ min. einmal vorkommen. Dieses Wort wird in der Messung gesucht und die Anzahl der Vorkommen zurückgegeben.

   - 1) 500
   - 2) 1.000
   - 3) 2.000
   - 4) 4.000
   - 5) 8.000
   - 6) 16.000
   - 7) 32.000
   - 8) 64.000
   - 9) 128.000

3. Resultate:

ADT-Hashmap									
	500	1000	2000	4000	8000	16000	32000	64000	128000
Laufzeit insert (ms)									
Laufzeit find (ms)									

Aus den Ergebnissen werden Excel-Graphiken erzeugt, um die Steigung der einzelnen interpolierten Kurven vergleichen zu können.

Die daraus resultierenden Schlussfolgerungen werden zusammen mit den Graphiken in einem PDF dokumentiert.