AD - Aufgabe 4 - Entwurf

Inhaltsverzeichnis

AD - Aufgabe 4 - Entwurf

Allgemeines

Team: 01 Eugen Deutsch, Phillip Schackier

Aufgabenaufteilung:
Die Aufgabe wurde zusammen bearbeitet.
Quellenangaben: AD Folien
Bearbeitungszeitraum:
02.01.16 - 5 Stunden: Beide

Aktueller Stand: Das Skript wurde fertiggestellt, sowie die Klasse HashMap
Änderungen in der Skizze: equals wurde beigefügt um die Tests zu vereinfachen.

Package-/Ordner-Hierarchie und Sichtbarkeiten

hash
----- Benchmark<public class>
----- HashMap<public class>
----- HashMapTime <public class>
----- tests
---------- HashMapTest<public class>

Klassen/interface Hierarchien und Sichtbarkeiten

Es gibt keinerlei Vererbungen untereinander.

Implementationen und Spezifikationen

Benchmark

- **public static void main(String** args**[])**: Wir lesen die beiden vorgegebenen Dateien ein und messen dabei die benötigte Zeit für ein insert und ein find in ns, wobei wir jeweils pro Datei 3 Messungen vornehmen (erster insert/find, mittlerer, letzter). Das machen wir jeweils für alle 3 Strategien, sodass wir 18 Messergebnisse für insert und 18 für find erhalten.

  Ausgegeben wird das Ergebnis in einer *.csv Datei.

  Außerdem wird über args der Dateiname und die Strategie eingegeben und die Anzahl der Vorkommen der einzelnen Wörter wird in einer log-Datei

(word_count.log) gespeichert, wobei dazu eine HashMap mit der gewählten Strategie benutzt wird. Sofern args leer oder fehlerhaft ist, wird einfach nur der Benchmark von insert und find durchgeführt.

HashMap

- **public static HashMap** create(**int** capacity**, Strategy** strategy**)**: Gibt eine leere HashMap mit der angegebenen Maximalgröße und Strategie zurück.

Bedingung/en: strategy != null, capacity > 0

- **public void** insert**(String** word**)**: Fügt ein Element in die HashMap ein und benutzt dafür folgendes Prinzip:

```
insert(String word)
    index = calcIndex(word)
    container[index] = word
    counters[index]++

Weitere Methoden:
    calcIndex(String word)
        k= hashCode(word)
        j = 0
        index = k

        while (container[index] != null && !container[index].equals(word))
            index = hashCode - hashProbe[strategy](j, k) % M
            j++

        return index

    hashCode(String word)
        int h = 0

        for (int i = 0; i < word.size; i++)
            h = (h * 128 + word[i]) mod M
        return h

    linearProbing(int j, int k)
        return j
```

AD - Aufgabe 4 - Entwurf

quadraticProbing(int j**, int** k**)**
return $(j / 2)^2 * (-1)^j$

doubleHashing(int j**, int** k**)**
return j * (1 + (k **mod** (M - 2)))

hashProbe = { **linearProbing**, **quadraticProbing**, **doubleHashing**}

Bedingung/en: word != null

- **public int** find**(String** word**)**: Gibt zurück, wie oft das gegebene Wort in der HashMap vertreten ist und benutzt dabei folgendes Prinzip:

find(String word**)**
index = calcIndex(word)
return counters[index]

- **public boolean** equals**(Object** obj**)**: Prüft die Gleichheit zwischen den Objekten und wird für die Tests benötigt.

HashMapTime

Ebenso wie **HashMap**, nur das hierbei die benötigte Zeit einer insert und find Operation in ns zurückgegeben wird.

Fehlerbehandlung

Wenn eine der Bedingungen nicht erfüllt wird, so wird der Algorithmus nicht ausgeführt und 0 oder null zurückgegeben. Es werden keinerlei Fehler geworfen.

Tests

Getestet wird die HashMap unter anderem in folgenden Fällen:

1. Fehlerhafte Eingaben
2. Gleiche Eingaben
3. Verschiedene Eingaben
4. Gleiche Reihenfolge der Eingaben, verschiedene Reihenfolge
5. Viele Eingaben
6. Alles jeweils mit allen 3 Strategien

AD - Aufgabe 4 - Entwurf

Export

Alles zusammen: hash.jar
Tests: hashTests.jar
Klassen: hashMapTime.jar, hashMap.jar, benchmark.jar