

Hash Tables – Separate Chaining Übung

Algorithmen und Datenstrukturen 2

- Grüne Farbe: Bitte im Script nachtragen

n|w

$$m = 10, n = 3 : \frac{10}{10} \cdot \frac{9}{10} \cdot \frac{8}{10} = \frac{10 \cdot 9 \cdot 8 \cdot \overbrace{7 \cdot 6 \cdot 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1}^{\text{?}}}{10 \cdot 10 \cdot 10 \cdot \overbrace{7 \cdot 6 \cdot 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1}^{\text{?}}}$$

1.) Wahrscheinlichkeit von Kollisionen

Wahrscheinlichkeit, dass es zu keinen Kollisionen kommt:

Anzahl Möglichkeiten, dass 1. Elem, 2. Elem ... n-Te Elem **ohne Kollisionen** eingefügt werden kann:

$$\frac{m \cdot (m - 1) \cdot (m - 2) \dots \cdot (m - (n - 1))}{m^n} = \frac{m!}{m^n \cdot (m - n)!}$$

Gegeben: Tabelle mit 13 Plätzen, 7 Elemente werden eingefügt.

Wahrscheinlichkeit für **keine** Kollision: $\frac{13!}{13^7 \cdot (6)!} \approx 0.138$

Wahrscheinlichkeit für eine Kollision: $1 - 0.138 \approx \mathbf{86 \% \text{ Wahrscheinlichkeit, für mind. 1 Kollision}}$

1.) Wahrscheinlichkeit von Kollisionen

Anzahl Kollisionen:

2

Max (Anzahl Kollisionen):

6

Durchschn. # Zugriffe:

$9 / 7 = 1.286$

Zahl x	x mod 13
1208	12
7351	6
9213	9
3298	9
2574	5
6832	7
5271	6

0	2574
1	
2	
3	
4	
5	
6	7351, 5271
7	6832
8	
9	9213, 3298
10	
11	
12	1208

2.) Berechnung von hashCode (1)

Wort	Hash-Wert
AUS	$65 + 85 + 83 = 233$
USA	$85 + 83 + 65 = 233$
SAU	$83 + 65 + 85 = 233$
OHR	$79 + 72 + 82 = 233$
WEM	$87 + 69 + 77 = 233$
OFT	$79 + 70 + 84 = 233$

3.) Berechnung von hashCode (2)

```
public int hashCode() {
    int h = hash;
    if (h == 0 && value.length > 0) {
        char val[] = value;
        for (int i = 0; i < value.length; i++) {
            h = 31 * h + val[i];
        }
        hash = h;
    }
    return h;
}
```

Wort	Hash-Wert
AUS	$31^2 \cdot 65 + 31 \cdot 85 + 83 = 65'183$
USA	$31^2 \cdot 85 + 31 \cdot 83 + 65 = 84'323$
SAU	$31^2 \cdot 83 + 31 \cdot 65 + 85 = 81'863$

$h = 0$

1.) $31 \cdot 0 + 65 = 65$

2.) $31 \cdot 65 + 85$

3.) $(31 \cdot 65 + 85) \cdot 31 + 83$

$\Rightarrow 31^2 \cdot 65 + 85 \cdot 31 + 83$

A: 65

U: 85

S: 83

4.) String.hashCode()

```
public int hashCode() {  
    int h = hash;  
    if (h == 0 && value.length > 0) {  
        char val[] = value;  
        for (int i = 0; i < value.length; i++) {  
            h = 31 * h + val[i];  
        }  
        hash = h;  
    }  
    return h;  
}
```



Hash-
Variable



Polynom
über 31



Horner-
Schema

$$a_0x^n + a_1x^{n-1} + a_2x^{n-2} + a_ix^{n-i} + \dots + a_n$$
$$= \left(\left((a_0 \cdot x + a_1) \cdot x + a_2 \right) \cdot x + \dots \right) \cdot x + a_n$$

5.) Fehlerhaftes Verhalten in hashCode() oder equals(Object o)

- Equals()
 - Logisch gleiche Objekte werden nicht als gleich gesehen
 - Equal-Vergleich schlägt fehl und Objekt wird deshalb nicht gefunden

MyInteger a = **new** MyInteger(7), b = **new** MyInteger(7);

System.out.println(a.equals(b)); // False

5.) Fehlerhaftes Verhalten in hashCode() oder equals(Object o)

- hashCode(): Logisch gleiche Objekte befinden sich an unterschiedlicher Stelle in der HashMap
- Deshalb: Immer hashCode() und equals() überschreiben!
 - Zuerst wird hashCode verglichen
 - Falls übereinstimmend, dann folgt ein equals-Vergleich


5.) Fehlerhaftes Verhalten in hashCode() oder equals(Object o) (2)

```
public boolean equals(Object other) {  
    return other instanceof MyInteger && ((MyInteger)other).i == i;  
}
```

```
public int hashCode() {  
    return i;  
}
```

6.) Personen-Klasse

```
public int hashCode() {  
    final int prime = 31;  
    int result = 1;  
    result = prime * result + birthday;  
    result = prime * result + birthmonth;  
    result = prime * result + birthyear;  
    result = prime * result + ((firstname == null) ? 0 : firstname.hashCode());  
    result = prime * result + ((lastname == null) ? 0 : lastname.hashCode());  
    return result;  
}
```



Final
Instanz-
Variablen

6.) Personen-Klasse, Java7 Variante

```
public int hashCode() {  
    return Objects.hash(firstname, lastname, birthyear, birthmonth,  
        birthday);  
}
```

6.) Personen-Klasse (2)

```
public boolean equals(Object obj) {  
    if (!(this.getClass() != obj.getClass())) {return false;}  
  
    Person other = (Person) obj;  
    if (birthday != other.birthday || birthmonth != other.birthmonth || birthyear !=  
        other.birthyear) {return false;}  
  
    if (firstname == null && other.firstname != null){ return false;}  
    else if (!firstname.equals(other.firstname)) {return false;}  
  
    if (lastname == null && other.lastname != null){return false;}  
    else if (!lastname.equals(other.lastname)){return false;}  
        return true;  
    }}
```

6.) Personen-Klasse (2), Java 7 Variante

@Override

```
public boolean equals(Object obj) {  
    if (this == obj) { return true; }  
    if (obj == null || this.getClass() != obj.getClass()) { return false; }  
    final Person other = (Person) obj;  
    return Objects.equals(this.firstname, other.firstname) &&  
        Objects.equals(this.lastname, other.lastname) &&  
        Objects.equals(this.birthyear, other.birthyear) &&  
        Objects.equals(this.birthmonth, other.birthmonth)&&  
        Objects.equals(this.birthday, other.birthday);  
}
```

8.) Korrekte Index-Berechnung

Ausgangslage (neg. Werte):

```
key.hashCode() % table.length;
```

Vorschlag (fehlerhaft):

```
Math.abs(key.hashCode()) % table.length;
```

Korrekte Vorschläge:

```
Math.abs(key.hashCode() % table.length);
```

```
(key.hashCode() & Integer.MAX_VALUE) % table.length;
```

9.) HashMap (Java 7)

Array-Initialisierung mit 2er Potenz:

```
int capacity = 1;
while (capacity < initialCapacity)
    capacity <<= 1;
table = new Entry[capacity];
```

Bit-Maskierung bei Index-Berechnung:

```
return hash & (capacity - 1);
```