Datenbanken, Hands-on 4

Henry Haustein

Aufgabe 1

Wir definieren dazu am Anfang des Quelltextes eine Variable `neuaufbau`, deren Wert wir auf 0 setzen. Weiterhin müssen wir den Wert dieser Variable immer dann um 1 erhöhen, wenn die Funktion `rehash_tables()` aufgerufen wird:

```
1 def rehash_tables(self) -> None:
2   global neuaufbau
3   neuaufbau = neuaufbau + 1
4   # create new hash functions
5   ...
```

Das Keyword `global` ist hier nötig, weil die Variable außerhalb der Funktion definiert wurde und damit normalerweise nicht veränderbar innerhalb der Funktion ist. Mit `global` kann man das ändern. Nach durchlaufen des Skripts hat `neuaufbau` dann den Wert von 841, das heißt die Hashtabellen wurden 841 mal neu aufgebaut.

Aufgabe 2

Hierfür legen wir uns ein Dictionary an, welches alle Sprachen und die Anzahl wie oft diese vertrieben worden sind, enthält:

```
1 counter = {sprache: 0 for sprache in languages_list}
```

Wir ändern dann einen Teil der `insert()`-Funktion:

```
1  tmp: Node = self.hash_table_1[self.hash_func_1(key)]
2  self.hash_table_1[self.hash_func_1(key)] = Node(key, value)
3  key = tmp.key
4  value = tmp.value
5  # sprache (value) wurde vertrieben -> counter erhoehen
6  counter[value] = counter[value] + 1
7
8  if self.hash_table_2[self.hash_func_2(key)] == None:
9  self.hash_table_2[self.hash_func_2(key)] = Node(key, value)
10 return True
11
12 # same as before for the other hash_table
13 tmp: Node = self.hash_table_2[self.hash_func_2(key)]
14 self.hash_table_2[self.hash_func_2(key)] = Node(key, value)
15 key = tmp.key
```

```
16  value = tmp.value
17  # sprache (value) wurde vertrieben -> counter erhoehen
18  counter[value] = counter[value] + 1
```

Ganz zum Schluss lassen wir dieses Dictionary noch sortieren mittels

```
1  print(sorted(counter.items(), key=lambda x: x[1], reverse=True))
```

und erhalten

```
1  [('C', 6812), ('Basic', 6377), ('C#', 5710), ('Ada', 5155), ('C++'
       , 4998), ('D', 4291), ('Kotlin', 4093), ('Perl', 3709), ('PHP'
       , 3609), ('Prolog', 3540), ('MATLAB', 3439), ('Javascript',
       3357), ('Eiffel', 3241), ('Pascal', 3193), ('Erlang', 3112), (
       'Fortran', 3109), ('Lisp', 2874), ('Go', 2808), ('Smalltalk',
       2594), ('Ruby', 2355), ('Scala', 2109), ('Haskell', 2055), ('
       Python', 2030), ('SQL', 1866), ('F#', 1732), ('Java', 1710), (
       'Swift', 1260), ('TypeScript', 228)]
```

Offensichtlich wurde C mit 6812 mal am öftesten aus dem Nest vertrieben.

Aufgabe 3

Die Löschfunktion ist

```
1  def delete(self, value):
2    # check hash table 1
3    for n in self.hash_table_1:
4      try:
5        if n.value == value:
6          self.hash_table_1[self.hash_func_1(n.key)] = None
7          break
8      except Exception as e:
9        pass
10
11   # check hash table 2
12   for n in self.hash_table_2:
13     try:
14       if n.value == value:
15         self.hash_table_2[self.hash_func_2(n.key)] = None
16         break
17     except Exception as e:
18       pass
```

Und dann noch

```
1  language_cuckoo.delete("Erlang")
2  language_cuckoo.delete("Python")
3  language_cuckoo.delete("TypeScript")
4  language_cuckoo.insert(8, "Erlang")
5  language_cuckoo.insert(27, "TypeScript")
6  language_cuckoo.insert(21, "Python")
7  language_cuckoo.print_hash_tables()
```

ergibt die folgenden Hashtables

Position	Hashtable 1	Hashtable 2
0	(0, Ada)	(1, Basic)
1		(13, Javascript)
2	(14, Kotlin)	
3	(24, Smalltalk)	
4	(2, C)	(12, Java)
5	(25, SQL)	
6	(16, MATLAB)	(10, Go)
7	(26, Swift)	(22, Ruby)
8	(4, C++)	(8, Erlang)
9	(27, TypeScript)	(21, Python)
10	(18, Perl)	(7, F#)
11	(9, Fortran)	(19, PHP)
12	(6, Eiffel)	(5, D)
13	(23, Scala)	(17, Pascal)
14	(20, Prolog)	(3, C#)
15	(11, Haskell)	(15, Lisp)

An der letzten Position der zweiten Hashtabelle steht also Lisp.

Aufgabe 4

Der Programmcode in (d) verläuft fälschlicherweise richtig. Problem ist hier, dass wir in der ersten Hashtabelle an der Position `hash2` nachschauen, obwohl die richtige Position hier `hash1` wäre. Selbiges für die 2. Hashtabelle.