

Algorithmen und Datenstrukturen SoSe25

-Assignment 7-

Moritz Ruge

Matrikelnummer: 5600961

Lennard Wittenberg

Matrikelnummer: —

Juni 2025

Problem 1: Hashing im Selbstversuch II

a) Fügen Sie nacheinander die Schlüssel 10, 22, 31, 4, 15, 28, 17, 88, 59 in eine Hashtabelle der Größe 11 ein. Die Hashfunktion sei $h(k) = k \bmod 11$. Die Konflikte werden durch offene Adressierung mit linearem Sondieren gelöst.

Lösung:

” ” : Empty, * : Deleted

Index	0	1	2	3	4	5	6	7	8	9	10
T =											

1. insert (10,v)

Index	0	1	2	3	4	5	6	7	8	9	10
T =											10

2. insert (22,v)

Index	0	1	2	3	4	5	6	7	8	9	10
T =	22										10

3. insert (31,v)

Index	0	1	2	3	4	5	6	7	8	9	10
T =	22									31	10

4. insert (4,v)

Index	0	1	2	3	4	5	6	7	8	9	10
T =	22				4					31	10

5. insert (15,v)

Index	0	1	2	3	4	5	6	7	8	9	10
T =	22				4	15				31	10

\Rightarrow index 4: $4 \neq 15 \Rightarrow$ Index 5: Empty \rightarrow Index 5 = 15

6. insert (28,v)

Index	0	1	2	3	4	5	6	7	8	9	10
T =	22				4	15	28			31	10

7. insert (17,v)

Index	0	1	2	3	4	5	6	7	8	9	10
T =	22				4	15	28	17		31	10

\Rightarrow Index 6: $28 \neq 17 \rightarrow$ Index 7: Empty \rightarrow Index 7 = 17

8. insert (88,v)

Index	0	1	2	3	4	5	6	7	8	9	10
T =	22	88			4	15	28	17		31	10

\Rightarrow Index 0: $22 \neq 88 \rightarrow$ Index 1: Empty \rightarrow Index 1 = 88

9. insert (59,v)

Index	0	1	2	3	4	5	6	7	8	9	10
T =	22	88			4	15	28	17	59	31	10

\Rightarrow Index 4: $4 \neq 59 \rightarrow$ Index 8: Empty \rightarrow Index 8 = 59

b) Fügen Sie nacheinander die Schlüssel 10, 22, 31, 4, 15, 29, 17, 88, 59 in eine Hashtabelle der Größe 11 ein. Die Konflikte werden durch Kuckuck gelöst, mit $h_1(k) = k \bmod 11$ und $h_2(k) = (k \bmod 13) \bmod 11$.

Illustrieren Sie jeweils die einzelnen Schritte.

Hinweis: Pseudocode für die Hashoperation findet sich im Skript.

Lösung:

” ” : Empty, * : Deleted

Index	0	1	2	3	4	5	6	7	8	9	10
T =											

1. insert (10)

$\Rightarrow h_1(k) = k \bmod 11 = 10 \bmod 11 = 10$

Index	0	1	2	3	4	5	6	7	8	9	10
T =											10

2. insert (22)

$\Rightarrow h_1(k) = k \bmod 11 = 22 \bmod 11 = 0$

Index	0	1	2	3	4	5	6	7	8	9	10
T =	22										10

3. insert (31)

$\Rightarrow h_1(k) = k \bmod 11 = 31 \bmod 11 = 9$

Index	0	1	2	3	4	5	6	7	8	9	10
T =	22									31	10

4. insert (4)

$\Rightarrow h_1(k) = k \bmod 11 = 4 \bmod 11 = 4$

Index	0	1	2	3	4	5	6	7	8	9	10
T =	22				4					31	10

5. insert (15)

$\Rightarrow h_1(15) = 15 \bmod 11 = 4$ Platziere in Index 4, $k' = 4$.
 $\rightarrow h_1(4') = 4 \bmod 11 = 4$. Anwendung $h_2(k')$
 $\rightarrow h_2(4') = (4 \bmod 13) \bmod 11 = 4$ Platz in Index 4, $k' = 15$.
 $\Rightarrow h_1(15') = 15 \bmod 11 = 4$. Anwendung $h_2(k')$
 $\rightarrow h_2(15') = (15 \bmod 13) \bmod 11 = 2$ Platz in Index 2.

Index	0	1	2	3	4	5	6	7	8	9	10
T =	22		15		4					31	10

6. insert (29)

$\Rightarrow h_1(k) = 29 \bmod 11 = 7$ Platziere in Index 7.

Index	0	1	2	3	4	5	6	7	8	9	10
T =	22		15		4			29		31	10

7. insert (17)

$\Rightarrow h_1(k) = 17 \bmod 11 = 6$ Platziere in Index 6.

Index	0	1	2	3	4	5	6	7	8	9	10
T =	22		15		4		17	29		31	10

8. insert (88)

$\Rightarrow h_1(88) = 88 \bmod 11 = 0$ Platziere in Index 0, $k' = 22$.
 $\rightarrow h_1(22') = 22 \bmod 11 = 0$. Anwendung $h_2(k')$
 $\rightarrow h_2(22') = (22 \bmod 13) \bmod 11 = 9$ Platz in Index 9, $k' = 31$.
 $\Rightarrow h_1(31') = 31 \bmod 11 = 9$ Anwendung $h_2(k')$
 $\rightarrow h_2(31') = (31 \bmod 13) \bmod 11 = 5$ Platz in Index 5 \rightarrow Empty.

Index	0	1	2	3	4	5	6	7	8	9	10
T =	88		15		4	31	17	29		22	10

9. insert (59)

$\Rightarrow h_1(59) = 59 \bmod 11 = 4$ Platziere in Index 4, $k' = 4$.
 $\rightarrow h_1(4') = 4 \bmod 11 = 4$. Anwendung $h_2(k')$
 $\rightarrow h_2(4') = (4 \bmod 13) \bmod 11 = 4$ Platz in Index 4, $k' = 59$.
 $\Rightarrow h_1(59') = 59 \bmod 11 = 4$ Anwendung $h_2(k')$.
 $\rightarrow h_2(59') = (59 \bmod 13) \bmod 11 = 7$ Platz in Index 7, $k' = 29$.
 $\Rightarrow h_1(29') = 29 \bmod 11 = 7$ Anwendung $h_2(k')$.
 $\rightarrow h_2(29') = (29 \bmod 13) \bmod 11 = 3$ Platz in Index 3 \rightarrow Empty.

Index	0	1	2	3	4	5	6	7	8	9	10
T =	88		15	29	4	31	17	59		22	10

Problem 2: Implementierung einer Hashtabelle

a) Implementieren Sie eine Hashtabelle mit Verkettung in Scala. Benutzen Sie dazu die Funktion `hashCode`, die von allen Objekten in Scala zur Verfügung gestellt wird.

Gestalten Sie Ihre Implementierung so, dass sich die Größe der Hashtabelle wählen lässt, und implementieren Sie mit mindestens zwei verschiedenen Kompressionsfunktionen.

b) Erweitern Sie Ihre Implementierung so, dass die Größe der Hashtabelle dynamisch angepasst wird, um einen Ladefaktor zwischen 1 und 3 zu garantieren (sobald mindestens 20 Einträge in der Hashtabelle vorhanden sind). Welche Strategie wählen Sie, um Ihre Hashtabelle anzupassen?

Datei: `Hashtable.scala`

```
1 class Hashtable(size : Int, compressionFunktion: Int => Int):
2   // public changable Attribute to alter, hold and display the (Key,Values) pairs.
3   // Using Array two a fixed size for the Hashtable and dynamic List Chains.
4   private var Table = Array.fill[List[(Int,Int)]](size)(Nil)
5   private var numEntries = 0 // counter pairs in Table
6   private var loadFactorCheck = 20 // indicator when the Table needs to be resized
7
8   private def resize(newSize : Int) : Unit =
9     val oldTable = this.Table
10    this.size = newSize
11    this.Table = Array.fill[List[(Int,Int)]](this.size)(Nil)
12    this.numEntries = 0
13    this.loadFactorCheck = loadFactorCheck * 2
14    // copy all old pairs into the resized Table.
15    for chain <- oldTable do
16      for (k,v) <- chain do
17        put(k,v)
18
19    // Two-Part hash-function 1. builtin hashCode and 2. class instances given
20    // compression function to create a index for the Table
21    def hash(key : Int) : Int =
22      val hashedPosition = key.hashCode()
23      val index = compression_division(hashedPosition,size)
24
25    // main 3 Operations
26    // get(k,v)
27    def get(key : Int) : Int =
28      val index : Int = hash(key) // 1. get the starting index
29      //2.Search the linked list at T[i] for an entry with key k
30      //3.If found, return the associated value
31      //4.If not found, throw NoSuchElementException in this case std error value
32      var chain : List[(Int,Int)] = this.Table(index)
33      var res : Int = -1 // std return value -> handling in main
34      for (k,v) <- chain do
35        if k == key then
36          res = v
37
38      res
39
40    //remove(k)
41    //1.Calculate index i = h(k)
42    //2.Search the linked list at T[i] for an entry with key k
43    //3.If found, remove that node from the linked list
44    //4.If not found -> nothing happens in this implementation
45    def remove(key : Int) : Unit =
46      val index : Int = hash(key)
47      val initalLength : Int = this.Table(index).length
48      Table(index) = Table(index).filter(_._1 != key)
```

```

47         if Table(index).length != initialLength then
48             this.numEntries -= 1 // keeping track of the number of entries
49
50         // put(k,v)
51         // 1.Calculate index i = h(k)
52         // 2.Search the linked list at T[i] for an entry with key k
53         // 3.If found, update its value to v
54         // 4.If not found, append a new node with (k,v) to the list at T[i]
55         def put(key : Int, value : Int): Unit =
56             val index : Int = hash(key)
57             var chain : List[(Int,Int)] = this.Table(index)
58             chain = chain.map(x => if(x._1 == key) then (key,value) else x) // map
59             over chain if entry.key == wanted key -> update value
60             if chain.filter(_._1 == key).length == 0 then // After
61                 mapping no entry with key present -> append
62                 chain = chain :: List((key, value))
63                 this.numEntries += 1 // keeping
64                 track of the number of entries
65                 this.Table(index) = chain // update
66                 the chain in Table with the new altered Chain
67             if numEntries >= loadFactorCheck then // if more Entries than the current
68                 loadFactorCheck allows are in the Table -> resize accordingly
69                 val loadFactor = this.numEntries.toDouble / this.size
70                 if loadFactor < 1 then // loadFactor too small
71                     resize(this.size/2)
72                 if loadFactor > 3 then // loadFactor too high
73                     resize(this.size*2)
74
75         //compression functions modeled after the script
76         def compression_division(hash : Int, size : Int): Int =
77             hash % size
78         def compression_multiplication(hash : Int, size : Int): Int =
79             val A : Float = 0.6180339887 // 0.6180339887 is the standard value from the script
80             .
81             val product = hash * A
82             val fraction = product - Math.floor(product)
83             (size * fraction).toInt
84
85         // main-function for debugging and testing
86         @main def main(): Unit =
87             var HT : HashTable = HashTable(10,compression_division)

```

Problem 3: Lineares Sondieren und Löschen

a) In der Vorlesung haben Sie eine Strategie gesehen, mit der das Löschen in einer Hashtabelle mit linearem Sondieren umgesetzt werden kann: Gelöschte Elemente werden durch einen eigenen Eintrag markiert und in den Einfüge- und Lookup Routinen speziell behandelt.

Beschreiben Sie eine alternative Methode, bei welcher der gelöschte Eintrag ggf. durch einen geeigneten anderen Eintrag ersetzt wird. Beschreiben sie Ihre Methode verbal und geben Sie Pseudocode. Geben Sie auch zwei interessante Beispiele, die zeigen, wie Ihre Methode funktioniert.

Lösung: Robin-Hood-Hashing mit Backwards-Shifting

Prinzip: Es handelt sich hierbei um eine Hashtabelle mit *Open Addressing*.

Beim Einfügen von neuen Schlüsseln wird für jeden Schlüssel der PSL (Probe Sequenz Lenght) berechnet um anzugeben, wie weit der Schlüssel von seiner eigentlichen Hashposition entfernt ist. Ziel ist es nun, das alle Schlüssel so nah wie möglich an ihrer "Home" Position bleiben. Beim Löschen von Schlüsseln wird geschaut ob es nachfolgend noch andere Schlüssel gibt, wenn nicht sind wir fertig, wenn doch wir auf deren PSL-Wert geschaut, ist dieser Größer als 0 Shiften wir den Schlüssel nach "Links" (auf die Position des gelöschten Schlüssels) und gegen in der Tabelle weiter, bis wir ein PSL-Wert von 0 oder einen Leeren Slot haben.

Einfügen:

⇒ insert: $h(8) = 1$

Index	0	1	2	3	4	5
T =		8 ₀				

⇒ insert: $h(20) = 1$

Index	0	1	2	3	4	5
T =		8 ₀				
		20 ₀ →				

Index	0	1	2	3	4	5
T =		8 ₀	20 ₁			

⇒ Wir verschieben 20 nach rechts und erhöhen den PSL um 1

⇒ insert: $h(3) = 1$

Index	0	1	2	3	4	5
T =		8 ₀	20 ₁			
		3 ₀ →	3 ₁ →			

Index	0	1	2	3	4	5
T =		8 ₀	20 ₁	3 ₂		

⇒ Wir verschieben 3 2x nach rechts und erhöhen den PSL um 2

⇒ insert: $h(9) = 0$

Index	0	1	2	3	4	5
T =	9 ₀	8 ₀	20 ₁			

⇒ insert: $h(66) = 0$

Index	0	1	2	3	4	5
T =	9 ₀	8 ₀	20 ₁			
	66 ₀ →					

Index	0	1	2	3	4	5
T =	9 ₀	66 ₁	20 ₁	8 ₂		
		8 ₀ →	8 ₁ →			

- ⇒ Wir verschieben 66 nach rechts und erhöhen den PSL um 1 → 66₁
 → Da 66₁ höher ist als 8₀ tauschen wir die Werte und verschieben die 8 nach rechts
 → Da 8₁ = 20₁ ist schieben wir den Schlüssel weiter nach rechts und erhöhen den PSL

Deletion:

⇒ delete: $h(9)$

Index	0	1	2	3	4	5
T =	9 ₀	8 ₀	20 ₁			
	↑					

→ Lookup 9

Index	0	1	2	3	4	5
T =		66 ₁	20 ₁	8 ₂		
	↑					

→ Lösche den Schlüssel

Index	0	1	2	3	4	5
T =		← 66 ₁	20 ₁	8 ₂		
		↑				

- ⇒ Schaue auf den folgenden Slot:
 → ist dieser leer sind wir fertig
 → ist dort ein Schlüssel, schauen wir auf den PSL ≥ 0 , Shiften wir den Eintrag nach links

Index	0	1	2	3	4	5
T =	66 ₀		← 20 ₁	8 ₂		
			↑			

→ PSL ≥ 0 , shift nach links

Index	0	1	2	3	4	5
T =	66 ₀	20 ₀		← 8 ₂		
				↑		

→ PSL ≥ 0 , shift nach links

Index	0	1	2	3	4	5
T =	66 ₀	20 ₀	8 ₁			
					↑	

→ Slot ist leer, Fertig!

Pseudocode: