

Department of Computer Science  
Markus Püschel  
Peter Widmayer  
Thomas Tschager  
Tobias Pröger

24th November 2016

## Algorithms & Data Structures

## Solutions to Sheet 9

## AS 16

### Solution 9.1 *Open Hashing.*

In the following, the probing methods are used exactly as they were presented in the lecture and in the book, i.e., the probing function is subtracted from the hash function. An addition would also be valid, but it might lead to different solutions.

- a)
- $h(k) = \text{Digit sum of } k$ . This function is not suitable for hashing. The value of the hash function must lie between 0 and  $p-1$ , but the digit sum can be arbitrarily large. Even if it was always below  $p$  it would not be a good choice because it does not lead to a uniform distribution of the keys.
  - $h(k) = k(1 + p^3) \bmod p$ . Since  $p^3 \bmod p = 0$  we have  $h(k) = k \bmod p$ , which is a suitable hashing function as explained in the lecture. The only drawback of the definition as stated in this exercise are the unnecessary arithmetic operations.
  - $h(k) = \lfloor p(rk - \lfloor rk \rfloor) \rfloor$ ,  $r \in \mathbb{R}^+ \setminus \mathbb{Q}$ . This is the so-called *multiplicative method* presented in the lecture, and it works well when  $r$  is chosen well. One can show that the method behaves particularly well for  $r = \phi^{-1} = \frac{\sqrt{5}-1}{2}$ .

- b) (i) 17:  $h(17) = 6$

						17					
--	--	--	--	--	--	----	--	--	--	--	--

6:  $h(6) = 6 \rightarrow h(6) - 1 \equiv 5$

					6	17					
--	--	--	--	--	---	----	--	--	--	--	--

5:  $h(5) = 5 \rightarrow h(5) - 1 \equiv 4$

				5	6	17					
--	--	--	--	---	---	----	--	--	--	--	--

8:  $h(8) = 8$

				5	6	17		8			
--	--	--	--	---	---	----	--	---	--	--	--

11:  $h(11) = 0$

11				5	6	17		8			
----	--	--	--	---	---	----	--	---	--	--	--

28:  $h(28) = 6 \rightarrow h(28) - 1 \equiv 5 \rightarrow h(28) - 2 \equiv 4 \rightarrow h(28) - 3 \equiv 3$

11			28	5	6	17		8			
----	--	--	----	---	---	----	--	---	--	--	--

14:  $h(14) = 3 \rightarrow h(14) - 1 \equiv 2$

11		14	28	5	6	17		8			
----	--	----	----	---	---	----	--	---	--	--	--

15:  $h(15) = 4 \rightarrow h(15) - 1 \equiv 3 \rightarrow h(15) - 2 \equiv 2 \rightarrow h(15) - 3 \equiv 1$

11	15	14	28	5	6	17		8			
----	----	----	----	---	---	----	--	---	--	--	--

Collisions: 9

(ii) 17:  $h(17) = 6$

						17				
--	--	--	--	--	--	----	--	--	--	--

6:  $h(6) = 6 \rightarrow h(6) - 1 \equiv 5$

					6	17				
--	--	--	--	--	---	----	--	--	--	--

5:  $h(5) = 5 \rightarrow h(5) - 1 \equiv 4$

				5	6	17				
--	--	--	--	---	---	----	--	--	--	--

8:  $h(8) = 8$

				5	6	17		8		
--	--	--	--	---	---	----	--	---	--	--

11:  $h(11) = 0$

11				5	6	17		8		
----	--	--	--	---	---	----	--	---	--	--

28:  $h(28) = 6 \rightarrow h(28) - 1 \equiv 5 \rightarrow h(28) + 1 \equiv 7$

11				5	6	17	28	8		
----	--	--	--	---	---	----	----	---	--	--

14:  $h(14) = 3$

11			14	5	6	17	28	8		
----	--	--	----	---	---	----	----	---	--	--

15:  $h(15) = 4 \rightarrow h(15) - 1 \equiv 3 \rightarrow h(15) + 1 \equiv 5 \rightarrow h(15) - 4 \equiv 0$   
 $\rightarrow h(15) + 4 \equiv 8 \rightarrow h(15) - 9 \equiv 6 \rightarrow h(15) + 9 \equiv 2$

11		15	14	5	6	17	28	8		
----	--	----	----	---	---	----	----	---	--	--

Collisions: 10

(iii) 17:  $h(17) = 6$

						17				
--	--	--	--	--	--	----	--	--	--	--

6:  $h(6) = 6 \rightarrow h(6) - h'(6) \equiv 10$

						17				6
--	--	--	--	--	--	----	--	--	--	---

5:  $h(5) = 5$

					5	17				6
--	--	--	--	--	---	----	--	--	--	---

8:  $h(8) = 8$

					5	17		8		6
--	--	--	--	--	---	----	--	---	--	---

11:  $h(11) = 0$

11					5	17		8		6
----	--	--	--	--	---	----	--	---	--	---

28:  $h(28) = 6 \rightarrow h(28) - h'(28) \equiv 4$

11				28	5	17		8		6
----	--	--	--	----	---	----	--	---	--	---

14:  $h(14) = 3$

11			14	28	5	17		8		6
----	--	--	----	----	---	----	--	---	--	---

15:  $h(15) = 4 \rightarrow h(15) - h'(15) \equiv 8 \rightarrow h(15) - 2h'(15) \equiv 1$

11	15		14	28	5	17		8		6
----	----	--	----	----	---	----	--	---	--	---

Collisions: 4

- c) The deletion of a key  $k$  is problematic if another key  $k'$  with  $h(k) = h(k')$  is inserted later than  $k$ . If  $k$  would simply be removed (e.g., by marking the position as free), the key  $k'$  could no longer be found, because the probing stops once an empty position is found. In the example above the keys 6 and 28 could no longer be found. Therefore we must mark the position explicitly as deleted, and when searching for a key the probing must continue when such a position is found. Of course, when inserting a new key such a marking may be overwritten if necessary.

If many keys are deleted, then it may happen that the search for a key gets very inefficient (because the probing may visit many positions that are marked as deleted). Therefore hashing is suitable especially if keys are mostly inserted and searched and only rarely deleted.

- d)
- $h'(k) = \lceil \ln(k+1) \rceil \bmod q$ . This function is not suitable as a second hash function, because for the key  $k = 0$  we have  $h'(0) = \lceil \ln(1) \rceil = 0$ .
  - $s(j, k) = k^j \bmod p$ . This function is not suitable as a probing function, because for the keys  $k = 0$  and  $k = 1$ , the function  $s(j, k)$  has constant value of 0 and 1.
  - $s(j, k) = ((k \cdot j) \bmod q) + 1$ . This function is also not suitable as a probing function because its value is constant 1 if the key  $k$  is a multiple of  $q$ . Moreover, for all other keys, the image of  $s(j, k)$  is  $\{1, \dots, q\}$ , i.e.,  $p - q$  addresses of the hash table cannot be reached.
- e) Quadratic probing generates a sequence of  $s(j, k)$  that does not depend on the key  $k$ . If there exist many keys  $k$  that are mapped on the same hash address  $h(k)$ , then the probing sequence is the same for all of these keys. Thus there are many collisions when probing. This phenomenon is called *secondary clustering*. Double hashing avoids secondary clustering because different keys  $k, k'$  with  $h(k) = h(k')$  often have different probing sequences.

### Solution 9.2 Cuckoo Hashing.

- a)
- Insertion of 27:
 

$T_1$ :			27		
---------	--	--	----	--	--

$T_2$ :					
---------	--	--	--	--	--
  - Insertion of 2 (replaces 27 in  $T_1$ ):
 

$T_1$ :			2		
---------	--	--	---	--	--

$T_2$ :	27				
---------	----	--	--	--	--
  - Insertion of 32 (replaces 2 in  $T_1$ , 2 replaces 27 in  $T_2$ , 27 replaces 32 in  $T_1$ ):
 

$T_1$ :			27		
---------	--	--	----	--	--

$T_2$ :	2	32			
---------	---	----	--	--	--
- b) Here we have multiple choices. For example, the insertion of key 7 leads to an endless loop.
- 1) Initially:
 

$T_1$ :			27		
---------	--	--	----	--	--

$T_2$ :	2	32			
---------	---	----	--	--	--
  - 2) Insert 7 into  $T_1$  (replaces 27):
 

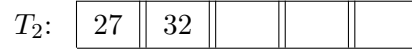
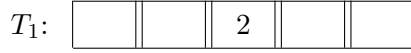
$T_1$ :			7		
---------	--	--	---	--	--

$T_2$ :	2	32			
---------	---	----	--	--	--
  - 3) Insert 27 into  $T_2$  (replaces 2):
 

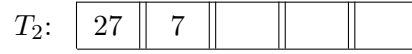
$T_1$ :			7		
---------	--	--	---	--	--

$T_2$ :	27	32			
---------	----	----	--	--	--

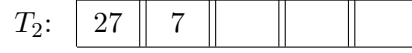
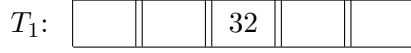
4) Insert 2 into  $T_1$  (replaces 7):



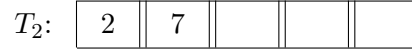
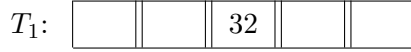
5) Insert 7 into  $T_2$  (replaces 32):



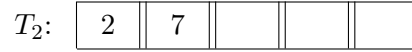
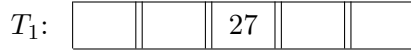
6) Insert 32 into  $T_1$  (replaces 2):



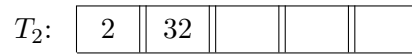
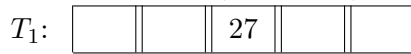
7) Insert 2 into  $T_2$  (replaces 27):



8) Insert 27 into  $T_1$  (replaces 32):



9) Insert 32 into  $T_2$  (replaces 7):



We have the identical configuration as initially, where the key 7 should be inserted in  $T_1$ .

**Solution 9.3** *Implementing a Queue with Stacks.*

As a reminder, amortized analysis using potential functions works as follows: We define  $\Phi_i$  as the potential for the  $i$ -th operation. Let the actual cost of the  $i$ -th operation be  $t_i$ . The *amortized cost* of the  $i$ -th operation is defined as  $a_i := t_i + \Phi_i - \Phi_{i-1}$ . With this definition, it follows for a sequence of  $m$  operations that

$$\sum_{i=1}^m a_i = \sum_{i=1}^m (t_i + \Phi_i - \Phi_{i-1}) = \left( \sum_{i=1}^m t_i \right) + \Phi_m - \Phi_0, \tag{1}$$

thus we obtain

$$\sum_{i=1}^m t_i = \sum_{i=1}^m a_i + \Phi_0 - \Phi_m. \tag{2}$$

Once we have an estimate of the amortized cost for each operation as well as an estimate for  $\Phi_0 - \Phi_m$ , we also have an estimate of the actual total costs. If the potential function is chosen such that  $\Phi_m \geq \Phi_0$  for every  $m$ , then it follows that  $\sum_{i=1}^m t_i \leq \sum_{i=1}^m a_i$ , i.e., the sum of the amortized costs is an upper bound for the actual total cost.

We denote the two stacks by  $\text{Stack}_L$  and  $\text{Stack}_R$ . Let them initially be empty. An `ENQUEUE` operation pushes the given object  $x$  on  $\text{Stack}_L$  (using the operation `PUSH`). The `DEQUEUE` returns the top element of  $\text{Stack}_R$  and removes it (with `POP`) if  $\text{Stack}_R$  is not empty. Otherwise, we iteratively remove the top element of  $\text{Stack}_L$  (with `POP`) and push it to  $\text{Stack}_R$  (with `PUSH`). We repeat this procedure until  $\text{Stack}_L$  is empty. Then, we return the top element of  $\text{Stack}_R$  and remove it from  $\text{Stack}_R$ .

We define the potential function  $\Phi_i$  as

$$2 \cdot \text{number of currently stored elements on } \text{Stack}_L. \tag{3}$$

An ENQUEUE operation has an actual cost  $t_i = 1$  and an amortized cost

$$a_i = t_i + \Phi_i - \Phi_{i-1} = 1 + 2 = 3. \quad (4)$$

In each DEQUEUE we first assume that  $\text{Stack}_R$  is not empty. Then, the actual and the amortized cost are both  $t_i = a_i = 1$ . Now let  $\text{Stack}_R$  be empty and  $k$  be the number of elements on  $\text{Stack}_L$ . The actual cost is  $t_i = 2k + 1$  in this case. The amortized cost is

$$a_i = t_i + \Phi_i - \Phi_{i-1} = 2k + 1 - 2k = 1. \quad (5)$$

Hence, the operations ENQUEUE and DEQUEUE can be computed in amortized running time  $\Theta(1)$ .