Institut für Pervasive Computing

Algorithms and Data Structures 2
Winter term 2024

Assignment 03

Deadline: **Wed. 04.12.2024, 23:59**

Time log: Please remember the time you needed to solve this assignment and log it in the feedback form in Moodle! This information is fully anonymous.

Hashing

Please, don't change any skeleton we provide. Of course, you are allowed to add code (methods and global variables), as long as you don't break the skeleton (method names and variables' types) we need for our tests to work correctly.

1. Chaining — 10 points

Implement the class `ChainingHashSet`, which internally uses a hash table for storing data. The size of a new hash table (i.e. the size of the list to which the hash values are mapped) has to be defined in the constructor. The hash table has to work according to the *chaining* principle and therefore store colliding elements in an overflow list.

The hash function *h* is defined as

$$h(k) = k \bmod N$$

where *k* is the key and *N* is the length of the hash table.

```
class ChainingHashSet():

    def get_hash_code(self, key):

    def get_hash_table(self):

    def set_hash_table(self, table):

    def get_table_size(self):

    def insert(self, key):

    def contains(self, key):

    def remove(self, key):

    def clear(self):

    def to_string(self):
```

For implementing the overflow list use nodes of the class `ChainingHashNode`. These nodes store a key and a reference to the next node in the chain.

```
class ChainingHashNode():
   def __init__(self, key = None):
      self.key = key
      self.next = None
```

Institut für Pervasive Computing

Assignment 03

Deadline: **Wed. 04.12.2024, 23:59**

2. Double Hashing — 10 points

Apply the **double hashing** algorithm presented in the exercise using **pen & paper**.

In contrast to example 1, this algorithm doesn't need additional memory for the insertion of colliding elements. In case of collisions no overflow list is used, instead the elements are stored at other vacant positions in the hash table using a 2nd hash function. The two hash functions *h1* and *h2* are defined as

h1(k) = k mod N
h2(k) = 1+ k mod (N-1)

where *k* is the key and *N* is the length of the hash table.

Calculate hash and offset as

hash = h1(k)
offset = h2(k)

Resolve any collisions by (repeatedly) applying the following probing sequence:

hash = h1(hash + offset)

a) Prefilling of table

Fill the following table using the provided insert statements:

For each digit also provide the probing sequence (indices separated by commas).

insert(32)

0	1	2	3	4	5	6	7	8	9	10	11	12

Probing sequence:

insert(18)

0	1	2	3	4	5	6	7	8	9	10	11	12

Probing sequence:

Institut für Pervasive Computing

Algorithms and Data Structures 2
Winter term 2024

Assignment 03

Deadline: **Wed. 04.12.2024, 23:59**

insert(24)

0	1	2	3	4	5	6	7	8	9	10	11	12

Probing sequence:

insert(39)

0	1	2	3	4	5	6	7	8	9	10	11	12

Probing sequence:

insert(27)

0	1	2	3	4	5	6	7	8	9	10	11	12

Probing sequence:

Institut für Pervasive Computing

Assignment 03

Deadline: **Wed. 04.12.2024, 23:59**

b) Insert of student ID

Fill the table, that is now prefilled with keys from part a), with the three rightmost digits of your student id. Again, note the probing sequence.
Please make a note in case one of the digits of your student id cannot be entered due to being a duplicate. Still write down the probing sequence that leads to the duplicate detection.

e.g.: student id = k12345**678** -> *insert(8), insert(7), insert(6)*

Student id: k X X X X X $\underset{d6}{\_\_\_}$ $\underset{d7}{\_\_\_}$ $\underset{d8}{\_\_\_}$

insert(d8)

0	1	2	3	4	5	6	7	8	9	10	11	12

Probing sequence:

insert(d7)

0	1	2	3	4	5	6	7	8	9	10	11	12

Probing sequence:

insert(d6)

0	1	2	3	4	5	6	7	8	9	10	11	12

Probing sequence:

Submission

Put your source files and a pdf for part 2 into a ZIP archive and name the file **k12345678-assignment03.zip** where k12345678 should reflect your student ID.