JOHANNES KEPLER UNIVERSITÄT LINZ

Algorithms and Data Structures 2
Winter term 2019
Stefan Grünberger, Dari Trendafilov

Assignment 07

Deadline: **Mon. 23.12.2019, 23:59**
Submission via: **www.pervasive.jku.at/Teaching/**

Elaboration time

*Remember the time you need for the elaboration of this assignment and document it in the file **time.txt** according to the structure illustrated in the right box. Please do not pack this file into an archive, but upload it as a **separate file**.*

```
#Student ID
k12345678
#Assignment number
07
#Time in minutes
190
```

Hashing

1. Chaining — 12 points

Implement the class *ChainingHashSet* based on the following interface *MyHashSet*

```
public interface MyHashSet {
    /**
     * returns the number of stored keys (these must be unique!).
     */
    public int size();

    /**
     * Inserts a key and returns true if it was successful.
     * If there is already an entry with the same key, the new key will
     * not be inserted and false is returned.
     */
    public boolean insert(Integer key, String data)
      throws IllegalArgumentException;

    /**
     *  Returns true if the key is already stored, otherwise false.
     */
    public boolean contains(Integer key)
      throws IllegalArgumentException;

    /*
     * Removes the key and returns true if it has been removed, false otherwise.
     */
    public boolean remove(Integer key)
      throws IllegalArgumentException;

    /*
     *  Returns a string representation of the hash table (array indices and stored element keys)
     *  Idx_0 {Node, Node, ... }, Idx_1 {...}
     *  0 {13}, 1 {82, 92, 12}, 2 {2, 32}, ...
     */
    public String toString();

    /*
     * Removes all stored elements
     */
    public void clear();

    /*
     * return the hashtable
     */
    public ChainingHashNode[] getHashTable();
}
```

and the abstract class `AbstractHashSet for calculating the hash code:`

```
abstract class AbstractHashSet{
         /*
          * calculates the hash code for a given key
          */
         public final int getHashCode(Integer key, Integer hashTableLength) {…};
}
```

Internally a hash table shall be used. The size of the hash table (i.e. the size of the array to which the hash values are mapped) has to be defined in the constructor (**`public ChainingHashSet(int capacity) {…}`**). The hash table has to work according to the *chaining* principle and therefore store colliding elements in an overflow list. Input parameters must not be `null`.

When you calculate the hash value use the Java method
`public int hashCode()`
(which is already defined in the class `Object`) and as hash function the **Modulo (%) operation**. For implementing the overflow list use the following *ChainingHashNode* class:

```
public class ChainingHashNode {
         Integer key;
         String data;
         ChainingHashNode next;

         ChainingHashNode(Integer key, String data) {
                  this.key = key;
                  this.data = data;
                  next = null;
         }
}
```

2. Quadratic probing 12 points

Implement (based on `MyHashSet` from example 1) the class `QuadraticHashSet`. This class should work according to the method of **quadratic probing** and therefore need no additional memory for the insertion of elements. In case of collisions no overflow list is used, these elements are stored at other vacant positions in the hash table.

The probing sequence for quadratic probing is defined as: h(k), h(k)+1, h(k)-1, h(k)+4, h(k)-4, ...

The first vacant position found is used for storing. For deleting the corresponding element is only marked as "deleted" (variable `removed` in class `OpenHashNode`), in order to avoid gaps (see exercise slides for more information).

Initialize the size of hash table using the constructor as in the first example (you don't have to implement dynamic resizing of the hash table!). For the hash function again use the Java method `hashCode()`. For the implementation of the array use the following class:

```
public class OpenHashNode {
     Integer key;
     String data;
     boolean removed = false;

     OpenHashNode(Integer key, String data) {
     this.key = key;
     this.data = data;
     removed = false;
     }
}
```