Data structures and algorithms

Project Documentation

ADT SET – implementation on a hash table, collision resolution by separate chaining

Farcasiu Razvan-Paul, group 913

# Problem statement

An IT company needs to store the data of all their employees. The application will use a set implemented on a hash table, where each key is unique and represents the telephone number of the employee.

Hashing provides O(1) time on average for insert, search and delete.

Hash function maps a big number(the telephone number) or string to a small integer that can be used as index in hash table. Using this will result in a so called Collision (two keys with the same index so that we will need to store them in in a linked list of records that have same hash function value).

**Advantages:**

- Hash table never fills up, we can always add more elements to the chain

- It is mostly used when it is unknown how many and how frequently keys may be inserted or deleted (in our case we don't know when an employee may be fired or hired, the company may double its number of exmployees in one month or simply fire most of them).

# ADT SET

## Domain:

S = {s|s is a set with elements of the type TElem}

I = {it|it is an iterator over a set with elements of type TElem }

## Interface:

* **init (s)**

description: creates a new, empty set.

pre: true

post: s ∈ S, s is an empty set.

* **destroy(smm)**

description: destroys a set

pre: s ∈ S

post: the set was destroyed (allocated memory was freed)

* **add(s, e)**

description: adds a new element into the set.

pre: s ∈ S, e ∈ TElem.

post: the element was added into the set, If s contains the element already, no change is made.

* **remove(s, e)**

description: deletes an element from the set.

pre: s ∈ S , e ∈ TElem.

post: the element was deleted from set (if it was in set, otherwise nothing will happen).

* **find(s, e)**

description: verifies if an element is in the set.

pre: s∈S , e ∈TElem

post: True if e (the element) is in the list, False otherwise.

* **iterator(s, it)**

description: returns an iterator for the set

pre: s∈ S

post: it ∈ I , it is an iterator over the set s

## Interface for iterator:

* **init(it, c)**

description: creates a new iterator for the set s

pre: c is a container

post: it ∈ I and it points to the first element in s if the set is not empty or it is not valid

* **getCurrent(it, e)**

description: returns the current element from the iterator

pre: it ∈ I, it is valid

post: e ∈TElem, e is the current element from it

* **next(it)**

description: moves the current element from the container to the next element or makes the iterator invalid if no elements are left

pre: it ∈ I, it is valid

post: the current element from it points to the next element from the set

* **valid(it)**

description: verifies if the iterator is valid

pre: it ∈ I

post: valid ← True, if it points to a valid element from the set, False otherwise

# Representation of the ADT and implementation in Pseudocode

## Representation:

**TElem:**

k: integer (the key of the node)

vl: integer (the value)

**TNode:**

elem: TElem

next: ↑ Node

**Set:**

T: ↑Node[] //an array of pointers to nodes

m: Integer

h: TFunction //the hash function

**Iterator S:**

current: ↑TNode

S: s