Documentation

ADT **Map** – implementation on a **hash table, collision resolution** by **coalesced chaining**.

❖ Problem statement:

A student implemented an agenda to keep track of the number of hours spent for studying per day during a month(30 days). The application offers the possibility to add, delete and check the amount of time spent for learning. In addition, it can compute the average of worked hours per week.

❖ Justification:

A map is a container where the elements are pairs <key, value>. Similarly the chosen problem uses <day, hours> pairs. Keys have to be unique in a map and so are the days in a month(integers from 1 to 30). Moreover, each key has one single associated value (amount of hours). The problem is suitable for this implementation because the hash table provides a virtually direct access to objects(number of hours spent for studying) based on a key (day of the month --unique Integer). In addition, each key is associated with a value and elements can be found, inserted, and removed using the integer index as an array index

❖ Domain of the ADT Map:

M = {m|m is a map with elements e = (k, v), where k ∈ TKey and v ∈ TValue}

❖ Interface of the ADT:

- subalgorithm init(m)

  descr: creates a new empty map
  pre: true
  post: m ∈ M, m is an empty map.

- subalgorithm destroy(m)

  descr: destroys a map
  pre: m ∈ M
  post: m was destroyed

- subalgorithm add(m, k, v)

  descr: add a new key-value pair to the map
  pre: m ∈ M, k ∈ TKey, v ∈ TValue
  post: m’ ∈ M, m’ = m ∪ < k, v >

- subalgorithm remove(m, k, v)

  descr: removes a pair with a given key from the map
  pre: m ∈ M, k ∈ TKey
  post: v ∈ TValue, where

  $$v \leftarrow \begin{cases} v', \text{ if } \exists < k, v' > \in m \text{ and } m' \in M,\ m' = m \setminus < k, v' > \\ \text{0 TValue, otherwise} \end{cases}$$

- function search(m, k, v)

  descr: searches for the value associated with a given key in the map
  pre: m ∈ M, k ∈ TKey
  post: v ∈ TValue, where

  $$v \leftarrow \begin{cases} v', \text{ if } \exists < k, v' > \in m \\ \text{0 TValue, otherwise} \end{cases}$$

- function getIterator(m, it)

  descr: returns an iterator for a map
  pre: m ∈ M
  post: it ∈ I, it is an iterator over m.

- function getSize(m)

  descr: returns the number of pairs from the map
  pre: m ∈ M
  post: size ← the number of pairs from m

- function getKeys(m, s)

  descr: returns the set of keys from the map
  pre: m ∈ M
  post: s ∈ S, s is the set of all keys from m

- function getValues(m, b)

  descr: returns a bag with all the values from the map
  pre: m ∈ M
  post: b ∈ B, b is the bag of all values from m

- function getPairs(m, s)

  descr: returns the set of pairs from the map
  pre: m ∈ M
  post: s ∈ S, s is the set of all pairs from m

❖ Interface Iterator:

- subalgorithm init(it, ht):

  descr: creates an iterator over the given hash table
  pre: list ϵ HT
  post: it ϵ I, it is an iterator over the given hash table

- subalgorithm destroy(it):

  descr: destroys the given iterator
  pre: it ϵ I
  post: it was destroyed

- function getCurrent(it):

  descr: returns the current element in the hash table
  pre: it ϵ I
  post: getCurrent <- the current node(pair<key,value>) in the table

- function isValid(it):

  descr: checks if the iterator is valid
  pre: it ϵ I
  post: isValid <- true if the iterator is still valid, false otherwise

- subalgorithm next(it):

  descr: moves the iterator to the next node in the hash table
  pre: it ϵ I
  post: it' ϵ I, it' is positioned on the next node in the hash table

❖ ADT Representation:

- Node:

  key: TKey

  value: TVal

- HashTable:

  T: Node[]

  next: Integer[]

  m: Integer

  firstFree: Integer

  h: TFunction

- Iterator:

  list: Hash Table

  currentPos: Integer