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 for odd/even days.

* **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

* **ADT Representation:**
* Pair:

key: TKey

value: TVal

* Map:

T: Pair[]

next: Integer[]

hashSize: Integer

firstFree: Integer

hashFunction: TFunction

* Iterator:

list: ↑ Map

currentPos: Integer

\*in case of complexities, instead of hashSize, the notation “m” will be used

* **Domain of the ADT Map:**

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

* **Interface of the ADT:**
* subalgorithm init(m)

descr: creates a new empty map

pre: true

post: m ∈ M, m is an empty map.

@initialize m.hashFunction

@initialize m.hashSize

@create m.T and m.next as arrays with m.hashSize positions

for i 🡨 0, hashSize-1 ex:

m.T[i] 🡨 make\_pair(-1,-1)

m.next[i] 🡨 -1

end-for

m.firstFree🡨0

end-init

Complexity: Θ(m)

* subalgorithm destroy(m)

descr: destroys a map

pre: m ∈ M

post: m was destroyed

end-subalgorithm

Complexity: Θ(1)

* subalgorithm add(m, k, v)

descr: add a new key-value pair to the map

pre: m ∈ M, k ∈ TKey, v ∈ TVal

post: m’ ∈ M, m’ = m ∪ < k, v >

p🡨make\_pair(k, v)

pos🡨map.hashFunction(k)

if m.T[pos] = make\_pair(-1,-1) then

m.T[m.hashFunction(k)] 🡨p

m.next[pos]🡨 -1

else

if m.firstFree = m.hashSize then

@no more storage space

end-if

current🡨pos

while m.next[current] !=-1 do

current🡨m.next[current]

end-while

m.T[m.firstFree]🡨p

m.next[m.firstFree]🡨-1

m.next[current]🡨m.firstFree

changeFirstFree(m)

end-if

end-subalgorithm

Complexity: Θ(1) on average, Θ(n) - worst case, Θ(1)-best case

* subalgorithm remove(m, k, v)

descr: removes a pair with a given key from the map

pre: m ∈ M, k ∈ TKey

post: v ∈ TVal, where v’, if ∃ < k, v’ >∈ m and m’ ∈ M, m’ = m\ < k, v’ >

v ←

0 TVal, otherwise

i ← m.hashFunction(k)

j ← -1

idx ← 0

while (idx < m.hashSize and j = -1) execute

if m.next[idx] = i then

j ← idx

else

idx ← idx + 1

end-if

end-while

while i ≠ -1 and [m.T[i]].first ≠ k execute

j ← i

i ← m.next[i]

end-while

if I = -1 then

@key does not exist

else

over ← false

while over ≠ true do

p ← m.next[i]

pp ← i

while p ≠ -1 and m.hashFunction(m.T[p]) ≠ i execute

pp ← p

p ← m.next[p]

end-while

if p = -1 then

over ← true

else

m.T[i] ← m.T[p]

j ← pp

i ← p

end-if

end while

if j ≠ -1 then

m.next[j] ← m.next[i]

end-if

m.T[i] ← -1

m.next[i] ← -1

if m.firstFree > i then

m.firstFree ← i

end-if

end-if

end-subalgorithm

Complexity: O(m^2)

* 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’, if ∃ < k, v’ >∈ m

v ←

0 TValue, otherwise

pos 🡨 m.hashFunction(k)

while pos ≠ -1 AND [m.T[pos]].first ≠ k ex:

pos🡨m.next[pos]

end-while

search🡨[m.T[pos]].second

end-search

Complexity: Best case: Θ(1)(when the searched key is on the position given by the hashFunction). O(m) in worst case( ∑1=m, k=from 1 to m), but on average Θ(1)( 1/m\*∑1 ∑P(1)E(1), k=from 1 to m).

* function getNext(m, pos)

descr: gets the value of m.next from the given position

pre: m ∈ M, pos is an integer

post: the value of next from the given position is returned

getNext 🡨 m.next[pos]

end-function

Complexity: Θ(1)

* function getHashSize(m, size)

descr: returns the size of m

pre: m ∈ M

post: size 🡨 the size of m

getHashSize 🡨m.hashSize

end-function

Complexity: Θ(1)

* function getAverage(m, day)

descr: returns the average of worked hours for even(day=2) or odd(day=1) days

pre: m ∈ M, day is integer

post: average of the days with the given property

sumE🡨0

sumO🡨0

kE🡨0

kO🡨0

for i🡨0,hashSize-1 do

if [m.T[i]].first % 2=1 AND search([m.T[i]].first) ≠-1 then

sumO🡨sumO+[m.T[i]].second

kO🡨kO+1

else

if [m.T[i]].first % 2=0 AND search([m.T[i]].first) ≠-1 then

sumE🡨sumE+[m.T[i]].second

kE🡨kE+1

end-if

end-for

if day=1 then

getAverage🡨 sumO/kO

else

getAverage🡨 sumE/kE

end-if

end-subalgorithm

Complexity: Θ(n)

* function getIterator(m, it)

descr: returns an iterator for a map

pre: m ∈ M

post: it ∈ I, it is an iterator over m

getIterator 🡨 ↑Iterator

end-function

Complexity: Θ(1)

* function getNumberOfPairs(m)

descr: returns the number of pairs from the map

pre: m ∈ M

post: nr ← the number of pairs from m

p = make\_pair(-1, -1)

nr🡨0

for i🡨0, m.hashSize – 1 do

if [m.T[i]].first ≠ p.first AND [m.T[i]].second ≠ p.second then

nr 🡨 nr+1

getNumberOfPairs 🡨 nr

end-function

Complexity: Θ(n)

* function operator[ ](m, pos)

descr: returns the pair(<TKey,TVal>) from the given position

pre: m ∈ Map, pos is an integer

post: the pair from the given position is returned

operator[ ] 🡨m.T[pos]

end-function

Complexity: Θ(1)

* subalgorithm changeFirstFree(m)

descr: change the the value of m.firstFree with the next first free position

pre: m ∈ Map

post: the value of m.firstFree is set to the next free position

m.firstFree ← m.firstFree + 1

while m.firstFree < m.hashSize and m.T[ht.firstFree] ≠ -1 execute

ht.firstFree ← ht.firstFree + 1

end-while

end-subalgorithm

Complexity: Θ(m)

* subalgorithm testMap(m)

descr: all the functionalities from class map are tested

pre: m ∈ Map

post: functions are tested, in case of incompatibilities, assertion errors are thrown

Map m{};

m.add(1, 3);

m.add(2, 4);

m.add(6, 6);

m.add(12, 8);

assert(m.getAverage(2) == 6);

m.add(62, 4);

m.add(82, 5);

assert(m.getAverage(1) == 3);

assert(m[1] == make\_pair(1, 3));

assert(m.getNumberOfPairs() == 6);

m.add(22, 9);

m.add(42, 2);

assert(m.getNumberOfPairs() == 8);

m.remove(1);

assert(m.getNumberOfPairs() == 7);

m.remove(22);

m.remove(82);

assert(m.getNumberOfPairs() == 5);

assert(m.search(12)==8);

assert(m.search(42) == 2);

m.remove(62);

assert(m.getNumberOfPairs() == 4);

assert(m.getNext(1) == -1);

m.add(7, 1);

m.add(17, 2);

m.add(37, 3);

m.add(57, 3);

m.remove(7);

m.remove(17);

m.remove(57);

end-subalgorithm

Complexity: Θ(1)

* **Interface Iterator:**
* subalgorithm init(it, m):

descr: creates an iterator over the given map

pre: list ϵ M

post: it ϵ I, it is an iterator over the given map

it.list 🡨m

it.currentPos 🡨0

end-subalgorithm

Complexity: Θ(1)

* subalgorithm destroy(it):

descr: destroys the given iterator

pre: it ϵ I

post: it was destroyed

end-subalgorithm

Complexity: Θ(1)

* function getCurrent(it):

descr: returns the current element in the map

pre: it ϵ I, it is valid

post: getCurrent <- the current pair (<key,value>) in the map

getCurrent 🡨[it.list]->m[it.currentPos]

end-function

Complexity: Θ(1)

* function isValid(it):

descr: checks if the iterator is valid

pre: it ϵ I, it is valid

post: isValid <- true if the iterator is still valid, false otherwise

if it.currentPos ≠ -1 AND it.currentPos < [it.list]->getHashSize(list)

isValid 🡨 true

else

isValid 🡨 false

end-if

end-function

Complexity: Θ(1)

* subalgorithm next(it):

descr: moves the iterator to the next node in the map

pre: it ϵ I

post: it’ ϵ I, it’ is positioned on the next node in the map

it.currentPos 🡨 it.currentPos + 1

currentPair 🡨 getCurrent

end-subalgorithm

Complexity: Θ(1)

* subalgorithm test(it):

{

Map m{};

m.add(1, 3);

m.add(2, 4);

m.add(3, 6);

UI ui{ m };

Iterator\* it = m.getIterator();

std::pair<int,int> p= m.getIterator()->getCurrent();

assert(p.first == -1);

assert(p.second == -1);

it->next();

assert(it->getCurrent().first == 1);

assert(it->getCurrent().second == 3);

assert(it->isValid() == true);

for (int i = 0; i < m.getHashSize(); i++)

it->next();

assert(it->isValid() == false);

}

* **Solution of the problem:**
* Subalgorithm add(m)

read key,value

add(key,val)

end-subalgorithm

Complexity: Θ(1)

* Subalgorithm removeKey(m)

read key

remove(key)

end-subalgorithm

Complexity: Θ(1)

* Subalgorithm searchKey(m)

read key

search(key)

end-subalgorithm

Complexity: Θ(1)

* Subalgorithm averageHours(m)

read day

av🡨getAverage(day)

print av

end-subalgorithm

Complexity: Θ(1)