Name: Lazăr Maria
Group: 914

Data structures and algorithms

ADT MultiMap – implementation on a hash table, collision resolution by open adressing.

Problem statement:

In a hospital there are some patients and each of them suffer by one or more diseases. Each patient is identified by their Personal Identification Code. The hospital needs an application to manage their patients together with their diseases. The application should allow to:

- ✓ Add a patient together with their disease
- ✓ Remove a patient's disease when he is does not suffer by that disease anymore
- ✓ Display all the patient's diseases

Justification:

Given the fact that the patients from the hospital are uniquely identified by a key, respectively their Personal Identification Code and for each patient we need to know the diseases he suffer, the MultiMap would be the most suitable ADT for the given problem statement.

Domain and interface:

- ADT MultiMap

  MM = {**mm** | **mm** is a map with elements e = (k, v), where k ∈ TKey and v ∈ TElem.

init(**mm**)
descr: creates a new empty multimap
pre: true
post: mm ∈ **MM**, is an empty multimap

destoy(**mm**)
descr: destroys a multimap
pre: mm ∈ **MM**
post: mm was destroyed

add(**mm, k, v**)
descr: adds a new key-value pair to the multimap
pre: mm ∈ **MM**, k ∈ **TKey**, v ∈ **TValue**

post: mm' ∈ **MM**, mm' = mm + <k, v>

remove(**mm, k, v**)
descr: removes a pair with a given key from the map
pre: mm ∈ **MM**, k ∈ **TKey**, v ∈ **TValue**
post: v' ∈ **TValue**, where

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

search(**mm, k, v**)
descr: searches for the value associated with a given key in the map
pre: mm ∈ **MM**, k ∈ **TKey**
post: vl ∈ **TContainer**, where

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

iterator(**mm, it**)
descr: returns an iterator for a map
pre: mm ∈ **MM**
post: it ∈ **Iterator**, it is an iterator over mm

size (**mm**)
descr: returns the number of pairs from the map
pre: mm ∈ **MM**
post: size ← the number of pairs from mm

- ADT MultiMap Iterator
  Iterator= {it | it is an multimap iterator}

init(**it, mm**)
descr: creates a new iterator for a multimap
pre: mm ∈ **MM**
post: i ∈ **Iterator**, and it points to the first element in mm if mm is not empty or it is not valid

getCurrent (**it, e**)

descr: returns the current element from the iterator
pre: it ∈ **Iterator,** it is valid
post: e ∈ **TElem**

next(**it**)
descr: moves the current element from the container to the next element or makes the iterator invalid if no elements are left
pre: it ∈ **Iterator**, it is valid
post: the current element from it points to the next element from the container

valid(**it**)
description: verifies if the iterator is valid
pre: it ∈ **Iterator**
post: valid = $\begin{cases} true, & \text{if points to a valid element from the container} \\ false, & \text{otherwise} \end{cases}$

Representation for *ADT* MultiM*ap*

-using a hash table, collision resolution by open addressing.

TElem
value: ***string***
deleted: ***bool***

MM
T: ***string[]*** {Array of ***strings*** inside the ***MM*** instance}
V: ***TElem[]*** {Array of ***TElem*** inside the ***MM*** instance}
m:***Integer*** {Number of slots inside the ***MM*** instance}
+hashFunction: ***TFunction*** {Hash function inside the ***MM*** instance}

Iterator
mm: ***MM*** {The MultiMap iterated}
current: ***Integer*** {The position of the current element}

Operations for multimap on hash table:

- Subalgorithm init(mm) is {Θ(m)}

```
mm.m<-@value
mm.T <-@array of strings with size mm.m
mm.V<-@array of TElem with size mm.m
for i<-0, mm.m, execute
        mm.T[i]=””
```

```
        [mm.V[i]].value<-””
        [mm.V[i]].deleted<-false
```

- Subalgorithm hashCodeFunction(k, i) is {Θ(1)}

```
k2<-0
v[10]={14, 15, 16, 17, 18, 19, 20, 21, 22, 23}
for  i<-0, i<13 execute
        c<-k[i]-’0’
        k2=k2+i*v[c]
hashCodeFunction<-k2
```

- Subalgorithm hashFunction(k, i) is {Θ(1)}

```
k2<-hashCodeFunction(k,i)
hashFunction<-(k2+i)%m
```

- Subalgorithm resize(mm) is {Θ(m)}

```
mm.m<-mm.m*2
newElems<-@array of TElem with size mm.m
newPos<-@array of string with size mm.m
for i<-0, mm.m/2 execute
        newElems[i].value<-mm.V[i].value
        newElems[i].deleted<-mm.V[i].deleted
        newPos[i]<-mm.T[i];
mm.V<-newElems;
mm.T<-newPos;
@dealocate newElems
@dealocate newPos
```

- Subalgorithm search(mm, key, v) is {O(1)}

```
i<-0
ok<- -1
pos<- mm.hashFunction(key, i)
if mm.V[pos].value=v and mm.T[pos]=key and mm.V[pos].deleted=false then
        ok<-pos
while i<mm.m and mm.T[pos]<>”” and ok=-1 execute
        if mm.V[pos].value and mm.T[pos]==key and mm.V[pos].deleted=false then
                ok<-pos
        pos<-mm.hashFunction(key, i)
```

```
        i<-i+1
    search<-ok
```

Complexity: O(1)

BC: When the element we search for is on the first position - Θ(1)

WC: When the element we search for is on the last position or it does not exist - Θ(m)

AC: Θ(1)

- Subalgorithm add(mm, key, v) is: {O(1)}

```
pos<-mm.hashFunction(key, 0)
i<-0
while i<mm.m and mm.T[pos]<>”” execute
        j<-1
        if mm.T[pos]<>”” and mm.V[pos].deleted=true then
                ok<-0
                p<-mm.hashFunction(key, 1)
                while mm.T[p]<>”” and j<mm.m execute
                        if mm.V[p].deleted=false then
                                ok<-1
                        i<-i+1
                        p<-mm.hashFunction(key, j);
                if ok=0 then
                        mm.T[pos]<-key
                        mm.V[pos].value<-value
                        mm.V[pos].deleted<-false
                        add<-true
        i<-i+1;
        pos<-mm.hashFunction(key, i)
if i=mm.m
        mm.resize
        while i<mm.m and mm.T[pos]<>””
                i<-i+1
                pos<-mm.hashFunction(key, i)
mm.T[pos]<-key
mm.V[pos].value<-value
mm.V[pos].deleted<-false
add<-true
```

Complexity: O(m)

BC: When the elemnt we search for is on the first position - Θ(1)

WC: When the element we search for is on the last position or it does not exist - Θ(m)

AC: Θ(1)

- Subalgorithm remove(mm, key, v) is {O(1)}

```
p<- mm.hashFunction(key, 0)
i<-0
while i<mm.m and mm.T[p]<>”” execute
        if mm.T[p]=key and mm.V[p].value=value then
                mm.V[p].deleted<-true
                remove<-true
        i<-i+1
        p<- mm.hashFunction(key, i)
remove<-false
```

Complexity: O(1)

BC: When the elemnt we search for is on the first position returned by hash function – {Θ(1)}

WC: When the element we search for is on the last position - Θ(m)

AC: Θ(1)

- Subalgorithm size(mm) is { Θ(m)}

```
nr<-0
for i<-0, mm.m execute
        if mm.T[i]<>”” and mm.V[i].deleted=false
                nr<-nr+1
size<-nr
```

- Subalgorithm iterator(mm) is {Θ(1)}

```
Iterator<-@iterator on mm
```

Operations for iterator on hash table:

- Subalgorithm init(it, mm) is { Θ(1) }

```
it.mm<-mm
it.current<-0
```

- Subalgorithm getCurrent(it) is { Θ(1) }

```
getCurrent<-[it.mm].V[it.current].value
```

- Subalgorithm getDeleted(it) is { Θ(1) }
  getCurrent<-[it.mm].V[it.current].deleted

- Subalgorithm getCurrentKey(it) is { Θ(1) }
  getCurrent<-[it.mm].T[it.current]

- Subalgorithm getDeleted(it) is { Θ(1) }
  getCurrent<-[it.mm].V[it.current].deleted;

- Subalgorithm next (it) is { Θ(1) }
  it.current<-it.current+1

- Subalgorithm valid(it) is { Θ(1) }
  if it.current<[it.mm].m then
  valid<-true
  else
  valid<-false

Operations Hospital:

- Subalgorithm init(h, mm, it) is { Θ(1) }
  h.mm<-mm
  h.mmIterator<-it

- Subalgorithm addDisease(h, CNP, d) is { Θ(1) }
  ok<-[h.mm].add(CNP, d)
  addDisease<-ok

- Subalgorithm removeDisease(h, CNP, d) is {O(m)}
  ok<-[h.mm].remove(CNP, d)
  removeDisease<-ok
  Complexity: O(m)
  BC: When the element we search for is on the first position – {Θ(1)}
  WC: When the element we search for is on the last position or it does not exist - Θ(m)
  AC: Θ(m)

- Subalgorithm hasSpecificDisease(h, CNP, d) is {O(m)}
  ok<-[h.mm].search(CNP, d)

hasSpecificDisease<-ok

Complexity: O(m)

BC: When the elemnt we search for is on the first position - Θ(1)

WC: When the element we search for is on the last position or it does not exist - Θ(m)

AC: Θ(m)

- Subalgorithm diseasesOfAPatient (h, CNP) is { Θ(m)}

mit<-@iterator over multimap
h.mmIterator<-mit
while [h.mmIterator].valid execute
pos<-[h.mmIterator].getCurrentKey
del<-[h.mmIterator].getDeleted
if pos=CNP and del=false then
result<-result+[h.mmIterator].getCurrent
[h.mmIterator].next
@dealocate mit
diseasesOfAPatient<-result

Tests:

```
void Tester::testHashFunction()
{
	MultiMap* mm = new MultiMap();
	int nr = mm->hashFunction("2970807042645", 1);
	assert(nr == 0);
}

void Tester::testResize()
{
	MultiMap* mm = new MultiMap();
	mm->resize();
	assert(mm->m = 169);
}

void Tester::testFunctions()
{
	MultiMap* mm = new MultiMap();

	int nr = mm->hashFunction("2970807042645", 1);
	assert(nr == 0);

	assert(mm->add("2970807042645", "disease1") == true);
	assert(mm->T[12] == "2970807042645");
	assert(mm->V[12].getValue() == "disease1");
	assert(mm->V[12].getDeleted() == false);
	assert(mm->add("2970807042645", "disease1") == false);
```

```
        assert(mm->search("2970807042645", "disease1") == 12);
        assert(mm->search("2970807021112", "disease2") == -1);

        assert(mm->size() == 1);

        assert(mm->remove("2970807021112", "disease2") == false);
        assert(mm->remove("2970807042645", "disease1") == true);

        assert(mm->add("2970807042645", "disease2") == true);
        assert(mm->add("2970807042645", "disease3") == true);
        assert(mm->add("2970807042645", "disease4") == true);
        assert(mm->search("2970807042645", "disease4") == 1);

        assert(mm->remove("2970807042645", "disease2") == true);
        assert(mm->remove("2970807042645", "disease3") == true);
        assert(mm->remove("2970807042645", "disease4") == true);

        assert(mm->add("2970807042645", "disease3") == true);

        assert(mm->add("2970807042645", "disease2") == true);
        assert(mm->add("2970807042645", "disease4") == true);

        assert(mm->remove("2970807042645", "disease2") == true);
        assert(mm->add("2970807042645", "disease4") == false);

        assert(mm->add("2970807042645", "disease1") == true);
        assert(mm->add("2970807042645", "disease2") == true);
        assert(mm->add("2970807042645", "disease5") == true);
        assert(mm->add("2970807042645", "disease6") == true);
        assert(mm->add("2970807042645", "disease7") == true);
        assert(mm->add("2970807042645", "disease8") == true);
        assert(mm->add("2970807042645", "disease9") == true);
        assert(mm->add("2970807042645", "disease10") == true);
        assert(mm->add("2970807042645", "disease11") == true);
        assert(mm->add("2970807042645", "disease12") == true);
        assert(mm->add("2970807042645", "disease13") == true);
        assert(mm->add("2970807042645", "disease14") == true);
        assert(mm->m == 26);
}

void Tester::testIterator()
{
        MultiMap* mm = new MultiMap();
        MultiMapIterator* mi;
        mi = mm->iterator();
        assert(mm->add("2970807042645", "disease1") == true);
        assert(mm->add("2970807042645", "disease2") == true);
        assert(mm->add("2970807042645", "disease3") == true);
        assert(mm->add("2970807042645", "disease4") == true);
        assert(mi->getCurrent()== "disease2");
        mi->next();
        assert(mi->getCurrent() == "disease3");
        assert(mi->getCurrentKey() == "2970807042645");
        assert(mi->getDeleted() == false);
        assert(mi->valid() == true);

}
```