Todoran Ana-Corina
Group 917
Task 16

Project documentation

Data structures and algorithms

Name: *Todoran Ana-Corina*

Group: *917*

Date: *JUNE 13, 2017*

Contents

Todoran Ana-Corina
Group 917
Task 16

1) *Task*

ADT Bag – implementation on a hash table, collision resolution by coalesced chaining

2) *ADT Specification:*

- **Bag** = { b | b – bag with elements of type TElement }

- **TElement** -> the general element in containers

  The interface of TElem contains the following operations:

  - assignment ($e_1 \leftarrow e_2$)
    - **pre**: $e_1, e_2 \in$ TElem
    - **post**: $e'_1 = e_2$
  - equality test ($e_1 = e_2$)
    - **pre**: $e_1, e_2 \in$ TElem
    - **post**:

$$\text{equal} = \begin{cases} \text{True, if e1 = e2} \\ \text{False, otherwise} \end{cases}$$

- **Iterator =** { it | it – iterator over Bag }

3) ADT Interface:

a. *Bag:*

- ***init(b):***
  - **pre**: True
  - **post**: b ∈ Bag
  - **throws**: - (None)
- ***destroy(b):***
  - **pre**: b ∈ Bag
  - **post**: b is destroyed
  - **throws**: - (None)
- ***add(b, e):***
  - **pre**: b ∈ Bag **,** e ∈ TElement
  - **post**: $b' \in Bag$, $b' = b + \{ e \}$
  - **throws**: - (None)
- ***remove(b, e):***
  - **pre**: b ∈ Bag **,** e ∈ TElement
  - **post**: $b' \in Bag$, $b' = b - \{ e \}$
  - **throws**: - (None)
- ***size(b):***
  - **pre**: b ∈ Bag
  - **post**: size = the number of elements in b
  - **throws**: - (None)
- ***search(b, e):***
  - **pre**: b ∈ Bag **,** e ∈ TElement
  - **post**:

$$search = \begin{cases} \text{True, if e is in b} \\ \text{False, otherwise} \end{cases}$$

  - **throws**: - (None)
- ***resize(b):***
  - **pre**: b ∈ Bag
  - **post**: $b' \in Bag$, rehash(b)
  - **throws**: - (None)

- ***rehash(b, elements):***
  - **pre**: b ∈ Bag**,** elements – list of Telem from container
  - **post**: b' - bag, $m' = 2 * m$
  - **throws**: - (None)
- ***iterator(b):***
  - **pre**: b ∈ Bag
  - **post**: it ∈ Iterator, it – iterator over b
  - **throws**: - (None)

b. ***Iteraror*:**

- ***init(it, b):***
  - **pre**: b ∈ Bag
  - **post**: it ∈ Iterator , it – iterator over b pointing to ”first element”
  - **throws**: - (None)
- ***next(it):***
  - **pre**: it ∈ Iterator, it is a valid iterator
  - **post**: it' - pointing to the next element
  - **throws**: - (None)
- ***valid(it):***
  - **pre**: it ∈ Iterator
  - **post**:

$$\text{valid(it)} = \begin{cases} \text{True, if it valid} \\ \text{False, otherwise} \end{cases}$$

  - **throws**: - (None)
- ***getCurrent(it, e):***
  - **pre**: it ∈ Iterator
  - **post**: e ∈ TElement, e – the current element pointed by it
  - **throws**: - (None)

Todoran Ana-Corina
Group 917
Task 16

4) ADT Representation:

a. Bag (Implemented on a hash table, collision resolution by coalesced chaining):

- length : Integer
- m (capacity) : Integer
- T : TElement[]
- next : Integer[]
- firstFree : Integer
- h : Tfunction

b. Iterator:

- b : ↑Bag
- currentPos : Integer

5) Pseudocode implementation:

a. Bag:

- ***Subalgorithm init(bag) is:***

```
@ bag.m <- 50
@ bag.length <- 0
@ bag.firstFree <- 0;
@ alocate(T)
@ alocate(next)
for i <- 0, m – 1 execute
    @ bag.T[i] <- NIl
    @ bag.next[i] <- -1
end_for
```

- ***end_Subalgorithm***

- ***Subalgorithm destroy(bag) is:***

```
@ free(T)
@ free(next)
```

- ***Subalgorithm size(bag) is:***

```
size <- bag.length
```

- ***end_Subalgorithm***

- ***Subalgorithm add(bag, e) is:***

```
if bag.length = bag.m then
    @ resize
@ hashCode <- hashFunction(e)
if bag.T[hashCode] = NIl then
    @ bag.T[hashCode] <- e
else
    @ bag.T[bag.firstFree] <- e
    @ i <- hashCode
```

- while bag.next[i] ≠ -1 execute
- @ i <- bag.next[i]
- end_while
- @ bag.next[i] = bag.firstFree
- while bag.T[bag.firstFree] ≠ NIl execute
- @ bag.firstFree <- bag.firstFree + 1
- end_while
- end_if
- bag.length <- bag.length + 1
- ***end_Subalgorithm***

- ***Subalgorithm remove(bag, e) is:***
- @ hashCode <- hashFunction(e)
- @ i <- hashCode
- @ previous <- -1
- @ nextPos <- bag.next[i]
- while bag.T[i] ≠ e execute
- @ previous <- i
- @ i <- nextPos
- @ nextPos <- bag.next[i]
- end_while
- if nextPos = -1 then
- @ bag.next[previous] <- -1
- @ bag.T[i] <- NIl
- if bag.firstFree > i then
- bag.firstFree <- i
- end_if
- else
- if nextPos = hashFunction(bag.T[nextPos]) then
- @ bag.next[previous] <- nextPos
- @ bag.T[i] <- NIl
- @ bag.next[i] = -1
- if bag.firstFree > i then
- bag.firstFree <- i
- end_if
- else
- @ bag.T[i] <- bag.T[nextPos]
- @ bag.next[i] <- bag.next[nextPos]
- @ bag.T[nextPos] <- NIl
- @ bag.next[nextPos] = -1
- if bag.firstFree > nextPos then
- bag.firstFree <- i
- end_if
- end_if
- bag.length <- bag.length - 1
- ***end_Subalgorithm***

Todoran Ana-Corina
Group 917
Task 16

- ***Subalgorithm search(bag, e) is:***
- `@ hashCode <- hashFunction(e)`
- `@ i <- hashCode`
- `if bag.T[i] = e then`
- `    search <- true`
- `end_if`
- `while bag.next[i] ≠ -1 execute`
- `    @ i <- bag.next[i]`
- `    if bag.T[i] = e then`
- `        search <- true`
- `    end_if`
- `end_while`
- `search <- false`
- ***end_Subalgorithm***

- ***Subalgorithm resize(bag) is:***
- `@ listBackup <- copyElements()`
- `@ bag.m <- 2 * bag.m`
- `@ bag.length <- 0`
- `@ bag.firstFree <- 0;`
- `@ free(T)`
- `@ free(next)`
- `@ alocate(T)`
- `@ alocate(next)`
- `for i <- 0, m – 1 execute`
- `   @ bag.T[i] <- NIl`
- `   @ bag.next[i] <- -1`
- `end_for`
- `rehash(bag, listBackup)`
- `@ free(listBackup)`
- ***end_Subalgorithm***

- ***Subalgorithm rehash(bag, elements) is:***
- `for i <- 0, m / 2 execute`
- `   @ add(bag, elements[i])`
- `end_for`
- ***end_Subalgorithm***

- ***Subalgorithm iterator(bag, it) is:***
- `@ size <- bag.length`
- ***end_Subalgorithm***

b. *Iterator:*

- ***Subalgorithm init(it, bag) is:***
- `@ it.b <- bag`
- `@ it.currentPos <- 0`
- ***end_Subalgorithm***

- ***Subalgorithm valid(it) is:***
- `if [bag].length = 0 then`
- `    valid <- false`
- `end_if`
- `if [bag].currentPos = [bag].m then`
- `    valid <- false`
- `end_if`
- `valid <- true`
- ***end_Subalgorithm***

- ***Subalgorithm next(it) is:***
- `while it.currentPos ≠ [it.bag].m && [it.bag].next[it.currentPos] = 0 && [it.bag].T[it.currentPos] = NIl execute`
- `    @ it.currentPos <- it.currentPos + 1`
- `end_while`
- ***end_Subalgorithm***

- ***Subalgorithm getCurrent(it) is:***
- `getCurrent <- [it.bag].T[it.currentPos]`
- ***end_Subalgorithm***

6) *Tests for the container:*

- ***Subalgorithm testConstructrors() is:***
- `bag1.init()`
- `@ bag1 <- bag1`
- `@ assert(size(bag1) = 0)`
- ***end_Subalgorithm***

- ***Subalgorithm testAddFunctions() is:***
- `bag1.init()`
- `@ bag1 <- bag1`
- `add(bag1, 3)`
- `add(bag1, 5)`
- `add(bag1, 6)`
- `add(bag1, 7)`
- `add(bag1, 13)`
- `@ assert(size(bag1) = 5)`
- ***end_Subalgorithm***

- ***Subalgorithm testSearchFunction() is:***
- `bag1.init()`
- `@ bag1 <- bag1`
- `add(bag1, 3)`
- `add(bag1, 5)`
- `add(bag1, 6)`
- `add(bag1, 7)`

- add(bag1, 13)
- @ assert(search(bag1, 6) = true)
- @ assert(search(bag1, 25) = false)
- ***end_Subalgorithm***

- ***Subalgorithm testRemoveFunction() is:***
- bag1.init()
- @ bag1 <- bag1
- add(bag1, 3)
- add(bag1, 5)
- add(bag1, 6)
- add(bag1, 7)
- add(bag1, 13)
- remove(bag1, 7)
- remove(bag1, 3)
- @ assert(size(bag1) = 3)
- ***end_Subalgorithm***

- ***Subalgorithm testIterator() is:***
- bag1.init()
- @ bag1 <- bag1
- add(bag1, 3)
- add(bag1, 13)
- add(bag1, 33)
- add(bag1, 23)
- add(bag1, 2)
- add(bag1, 43)
- iterator(bag1, it)
- @ assert(valid(it) == false)
- init(bag1, it)
- while valid(it) execute
-     @ print getCurrent(it)
-     next(it)
- end_while
- ***end_Subalgorithm***

7) Operations complexity:

a. Bag

add

Best case - ϴ(1), Bag is empty

O(1) average case, amortized

Worst case O(n) when all n added elements are on the same chaining

Remove

Best case - ϴ(1), Bag is empty

O(1) average case, amortized

Worst case O(n') where n' is the chaing length of elements in which our elements is part of it

size - **ϴ(1)**

search

Best case - ϴ(1), Bag is empty

O(1) average case, amortized

Worst case O(n') when all our element is the last in it's chaining

iterator - **ϴ(m)**, where m is the capacity, from *Iterator init*

b. Iterator

init - **ϴ(m)**, where m is the capacity

next

O(1) average case, amortized

Worst case O(n) when all n added elements have the same hashFunction results

valid - **ϴ(1)**

getCurrent - **ϴ(1)**

8) Problem Statement:

Having a password database, memorised by it's password code, check if a given password exists. Password format consisting in numbers (each number has to have at least 4 digits and maximum 10). A password is not necessarily unique, so for the given password if it exists show also the number of apparitions.

Todoran Ana-Corina
Group 917
Task 16

9) Problem description:

We have an initial text file ”data.txt” in which is stored our password database. In order to solve this problem, we read a number and try to see if it is in our input database. If it is we cycle through the bag in O(n) in order to find the apparition frequency. Thus, the total complexity of the algorithm for search is O(n), and for getting the frequency is n * 1 i.e. O(m), where n is the number of elements from our Bag.

So overall complexity is O(m+n) = O(m). (n < m)

10) Problem solution:

- ***Subalgorithm searchPassword(bag, number) is:***
- `searchPassword <- bag.search(number)`
- ***end_Subalgorithm***

- ***Subalgorithm getFrequency(bag, number) is:***

```
if nextPos = -1 then
    init(it)
    @ fr <- 0
    @ e <- number
    while valid(it) execute
        if getCurrent(it) = e then
            fr <- fr + 1
        end_if
        next(it)
    end_while
else
    getGrequency <- 0
end_if
```

- ***end_Subalgorithm***

- ***Subalgorithm SolveProblem(bag, number) is:***

```
@ read number
find <- searchPassword(number)
if find = true then
    @ print Password was found in database!
    @ fr <- getFrequency(number)
    @ print Password frequency is: fr
else
    @ print Inexistent password!
end_if
```

- ***end_Subalgorithm***

11) *Solution complexity:*

a. ***Search***:

- **Best case - ϴ(1),** Bag is empty, so ne cannot find our password
- **O(1) average case, amortized**
- **Worst case O(n')** when our password is the end of it's chaining

b. ***Get Frequency***:

- **Best case** = O(1), the password is not in our database
- **Worst case = Average Case = ϴ(n)** because all the time we iterate through the hole Bag