SDA - Seminar 1

Date de contact:
Email : marianzsu@cs.ubbcluj.ro
Pagina : www.cs.ubbcluj.ro/~marianzsu

TAD Colecție

- Ca o mulțime, dar elementele nu sunt distincte
- Nu contează ordinea elementelor
  - Nu există poziții într-o colecție – operațiile colecției nu primesc poziție ca parametru
  - Elementele adăugate nu sunt stocate în ordinea adăugării (pot fi, dar nu e garantat că se întâmplă așa)

De exemplu:

- Adăugăm în Colecție elementele 1,3,2,6,2,5,2
- Dacă afișăm conținutul colecției, orice ordine a elementelor e posibilă:
  - 1, 2, 2, 2, 3, 6, 5
  - 1, 3, 2, 6, 2, 5, 2
  - 1, 5, 6, 2, 3, 2, 2
  - Etc...

Domeniu: $\mathcal{C}$ = {c | c este o colecție cu elemente de tip TElement}

Interfața (operații):

creeaza(c)
pre : true
post: c ∈ $\mathcal{C}$, c este colecția vidă

adaugă(c, e)
pre: c ∈ $\mathcal{C}$, e – Telement
post: c' ∈ $\mathcal{C}$, c' = c U {e}

sterge(c, e)
pre: c ∈ $\mathcal{C}$, e – Telement
post: c' ∈ $\mathcal{C}$, c' = c \ {e}

cauta(c, e)
pre: c ∈ $\mathcal{C}$, e – Telement
post: $cauta \leftarrow \begin{cases} adevarat, daca\ e \in \mathbf{C} \\ false, altfel \end{cases}$

dim(c)
pre: c ∈ $\mathcal{C}$
post: dim ← $nr\ de\ elemente\ din\ c$

distruge(c)

pre: c ∈ C

post: c a fost distrus

iterator(c, i)

pre: c ∈ C

post: i ∈ I, i este iterator pe c

TAD Iterator

- Are acces la structura interioară a colecției și un element curent din colecție

Domeniu: I = {i | i este iterator pe c ∈ C }

Interfața:

creeaza (i, c)

pre: c ∈ C

post: i ∈ I, i este iterator pe c

valid(i)

pre: i ∈ I

post: $valid \leftarrow \begin{cases} adevarat, dacă\ elementul\ curent\ din\ i\ este\ un\ element\ valid \\ fals, altfel \end{cases}$

urmator(i)

pre: i ∈ I

post: i' ∈ I , elementul curent din i' referă următorul element din c

element(i, e)

pre: i ∈ I

post: e – Telement, e este elementul curent din iterator

Exemple pentru iterator:

- Python
  - for x in container:
  - dacă vreți să implementați iterator
    - __iter__(self)
    - next(self)
      - returnează elementul curent
      - trece la următorul element
      - ridică *StopIteration* exception când nu mai este valid
- C++
  - STL containers
  - begin(), end(), ++, --
- Java
  - Containerele au operație *iterator()*
  - hasNext()
  - next()
    - returnează elementul curent

- trece la următorul element

Reprezentare:

1. - înșiruire de elemente care se pot repeta
   Iteratorul conține poziția/indexul elementului curent

1	3	2	6	2	5	2

2. perechi: element + frecvență (vector de perechi)
   Iteratorul are un element curent și o frecvență curentă pentru elementul curent

(1,1)	(3,1)	(2,3)	(5,1)	(6,1)

Implementare Python

```
class Colectie:

    def __init__(self):
        #operatia creeaza
        self.__c = []

    def adauga (self, element):
        self.__c.append(element)

    def sterge (self, element):
        if element in self.__c:
            self.__c.remove(element)

    def cauta (self, element):
        if element in self.__c:
            return True
        return False

    def dim (self):
        return len(self.__c)

    def iterator (self):
        return Iterator(self)


class Iterator:

    def __init__(self, colectie):
        self.__colectie = colectie
        self.__curent = 0

    def prim (self):
        self.__curent = 0

    def element(self):
        #operatia element trebuie sa acceseze lista din colectie, dar ea nu
       #poate fi accesata direct, pentru ca este private. De aceea folosim aceasta
```

```
         #metoda de a accesa lista.
        return self.__colectie._Colectie__c[self.__curent]

    def urmator(self):
        self.__curent = self.__curent + 1

    def valid(self):
        return self.__curent < self.__colectie.dim()
```

Program principal

```
def creeazaColectieIntregi(colectie):
    colectie.adauga(1)
    colectie.adauga(2)
    colectie.adauga(3)
    colectie.adauga(2)
    colectie.adauga(3)
    colectie.adauga(4)
    colectie.adauga(1)
    colectie.adauga(3)

def creeazaColectieString(colectie):
    colectie.adauga("abc")
    colectie.adauga("bcd")
    colectie.adauga("abd")
    colectie.adauga("abc")
    colectie.adauga("xyz")
    colectie.adauga("abc")

def tipareste(colectie):
    iterator = colectie.iterator()
    while iterator.valid():
        print (iterator.element())
        iterator.urmator()


def main():
    cI = ColectieF()
    creeazaColectieIntregi(cI)
    tipareste(cI)

    cS = Colectie()
    creeazaColectieString(cS)
    tipareste(cS)
```

Colectie cu frecventa

```
class ElementPereche:

    def __init__(self, element, frecv):
        self.__element = element
        self.__frecv = frecv

    def getElement(self):
        return self.__element

    def getFrecventa(self):
        return self.__frecv

    def setFrecventa(self, fr):
        self.__frecv = fr

    def __eq__(self, other):
        return other.__element == self.__element

class ColectieF:

    def __init__(self):
        self.__colectie = []

    def adauga(self, element):
        gasit = False
        for elem in self.__colectie:
            if elem.getElement() == element:
                elem.setFrecventa(elem.getFrecventa() + 1)
                gasit = True
                break
        if not gasit:
            elem = ElementPereche(element, 1)
            self.__colectie.append(elem)

    def sterge(self, element):
        for elem in self.__colectie:
            if elem.getElement() == element:
                if elem.getFrecventa() > 1:
                    elem.setFrecventa(elem.getFrecventa() - 1)
                else:
                    e = ElementPereche(element, 1)
                    self.__colectie.remove(e)

    def cauta(self, element):
        for elem in self.__colectie:
            if elem.getElemet() == element:
                return True
        return False

    def dim(self):
        numar = 0
        for elem in self.__colectie:
            numar = numar + elem.getFrecventa()
        return numar
```

```
    def iterator(self):
        return IteratorF(self)


class IteratorF:
    def __init__(self, colectie):
        self.__colectie = colectie
        self.__elemCurent = 0
        self.__frecvCurenta = 0

    def prim(self):
        self.__elemCurent = 0
        self.__frecvCurenta = 0

    def urmator(self):
        #accesam elementul (perechea) curenta din colectie
        e = self.__colectie._ColectieF__colectie[self.__elemCurent]
        if e.getFrecventa() - 1 > self.__frecvCurenta:
            self.__frecvCurenta = self.__frecvCurenta + 1
        else:
            self.__elemCurent = self.__elemCurent + 1
            self.__frecvCurenta = 0

    def element(self):
        e = self.__colectie._ColectieF__colectie[self.__elemCurent]
        return e.getElement()

    def valid(self):
        return self.__elemCurent < len(self.__colectie._ColectieF__colectie)
```