SDA - Seminar - TAD Colecție

- **Cuprins**:
  - Convenții
  - TAD Colecție - definire și specificare
  - TAD Iterator - specificare
  - Exemplificare reprezentare
  - Implementare Python

Convenții

- Algoritmii vor fi descriși în **Pseudocod**
- Elementele din containere vor fi elemente generale/abstracte
  - **TElement**
    - Operație de atribuire: $e_1 \leftarrow e_2$
    - Verificarea egalității: $e_1 = e_2$
  - **TComparabil**
    - Pentru containere ordonate
    - Permite inclusiv aplicarea unor relații de ordine: <, >, etc

TAD Colecție

Definire

- Spre deosebire de cazul mulțimii, elementele nu sunt neapărat distincte, putându-se, deci, repeta
- Precum și în cazul mulțimii, ordinea elementelor este irelevantă
  - Nu există poziții într-o colecție. Așadar, operațiile colecției nu primesc poziții ca parametri și nu returnează poziții.
  - Elementele adăugate nu sunt stocate în ordinea adăugării (cel puțin, nu avem această garanție)

    De exemplu, dacă adăugăm într-o colecție elementele 1, 3, 2, 6, 2, 5, 2, la afișarea conținutului colecției i, 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...

Specificare

1) Definirea domeniului:

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

2) Specificarea interfeței:

creeaza(c)

pre : adevărat

post: c ∈ $\mathcal{C}$, c este colecția vidă (sau dim(c) = 0)

distruge(c)

pre: c ∈ $\mathcal{C}$

post: colecția c a fost distrusă

adauga(c, e)

pre: c ∈ $\mathcal{C}$, e: TElementpost: c' ∈ $\mathcal{C}$, c' = c U {e} (sau c' = c $\oplus$ {e})

sterge(c, e)

pre: c ∈ $\mathcal{C}$, e: TElement

post: c' ∈ $\mathcal{C}$, c' = c \ {e} (**OBS**: se șterge o singură apariție a elementului e)

cauta(c, e)

pre: c ∈ $\mathcal{C}$, e: TElement

post: $cauta = \begin{cases} \text{adevărat, dacă e} \in \text{c} \\ \text{fals, altfel} \end{cases}$

dim(c)

pre: c ∈ $\mathcal{C}$

post: $\text{dim} = \text{numărul total de elemente din c}$

iterator(c, i)

pre: c ∈ $\mathcal{C}$

post: i ∈ $\mathcal{I}$, i este iterator pe c și i referă un prim element din c

TAD Iterator

Specificare

1) Definirea domeniului:

$\mathcal{I}$ = {i | i este iterator pe c ∈ $\mathcal{C}$ }

2) Specificarea interfeței:

creeaza (i, c)

pre: c ∈ $\mathcal{C}$

post: i ∈ $\mathcal{I}$, i este iterator pe colecția c și referă un prim element din c

valid(i)

pre: i ∈ $\mathcal{I}$

post: $valid = \begin{cases} adevarat, \text{dacă elementul curent referit de i este valid} \\ fals, \text{altfel} \end{cases}$

urmator(i)

pre: i ∈ $\mathcal{I}$

post: i' ∈ $\mathcal{I}$, i' referă următorul element din colecția iterată față de cel referit de i

element(i, e)

pre: i ∈ $\mathcal{I}$

post: e: Telement, e este elementul curent referit de iterator

Exemplificare reprezentare:

O primă variantă de reprezentare (R1):

- Ca tablou unidimensional (vector) de elemente care se pot repeta
- În acest caz, iteratorul conține indexul elementului curent

Exemplu:

1	3	2	6	2	5	2

O a doua variantă de reprezentare (R2):

- Ca vector de perechi (element, frecventa), elementele din perechi fiind distincte
- În acest caz, nu mai este suficient ca în reprezentarea iteratorului să avem doar indexul curent. Este necesară și frecvența curentă pentru elementul de la indexul curent.

Exemplu:

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

Implementare Python

- Reprezentare R1:

```
class Colectie:
    def __init__(self):
        self.__elemente = []
    def adauga(self, e):
        self.__elemente.append(e)
    def cauta(self, e):
        return e in self.__elemente
    def sterge(self, e):
        return self.__elemente.remove(e)
    def dim(self):
        return len(self.__elemente)
    def iterator(self):
        return Iterator(self)

class Iterator:
    def __init__(self,c):
        self.__c = c
        self.__curent = 0
    def valid(self):
        return self.__curent < self.__c.dim()
    def element(self):
        return self.__c._Colectie__elemente[self.__curent]
    def urmator(self):
        self.__curent = self.__curent+1

def populeazaColectieIntregi(c):
    c.adauga(1)
    c.adauga(2)
    c.adauga(3)
    c.adauga(2)

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

def main():
    c = Colectie()
    populeazaColectieIntregi(c)
    tipareste(c)
```

- Reprezentare R2:

```
class PerecheElementFrecventa:
```

```
2.      def __init__(self,e,f):
3.          self.__e = e
4.          self.__f = f
5.      def setElement(self, e):
6.          self.__e = e
7.      def setFrecventa(self, f):
8.          self.__f = f
9.      def getElement(self):
10.         return self.__e
11.     def getFrecventa(self):
12.         return self.__f
13.     def __eq__(self, other):
14.         return (self.__e == other.__e)
15.
16. class ColectieElementFrecventa:
17.
18.     def __init__(self):
19.         self.__perechi = []
20.
21.     def adauga(self, e):
22.     #Distingem doua cazuri: elementul apare sau nu deja in colectie
23.         # Daca elementul apare deja in colectie, ii incrementam frecventa
24.         # Daca elementul nu apare deja in colectie, il adaugam, cu frecvent 1
25.         gasit = False
26.         for p in self.__perechi:
27.             if p.getElement()== e:
28.                 p.setFrecventa(p.getFrecventa()+1)
29.                 gasit = True
30.         if not(gasit):
31.             p = PerecheElementFrecventa(e,1)
32.             self.__perechi.append(p)
33.
34.     def cauta(self, e):
35.          for p in self.__perechi:
36.             if p.getElement() == e:
37.                 return True
38.          return False
39.
40.     def sterge(self, e):
41.     #Distingem trei cazuri: elementul nu arape in colectie, elementul apare in co
    lectie cu o singura data sau elementul apare in colectie de mai multe ori
42.         # Daca elementul nu apare in colectie, colectia ramane nemodificata
43.         # Daca elementul apare in colectie o singura data, perechea care il conti
    ne va fi stearsa
44.         # Daca elementul apare in colectie de mai multe ori, frecventa lui va fi
    decrementata
45.         for p in self.__perechi:
46.             if p.getElement() == e:
47.                 if p.getFrecventa() > 1:
48.                     p.setFrecventa(p.getFrecventa()-1)
```

```
                else:
                    self.__perechi.remove(p)

    def dim(self):
        d = 0
        for p in self.__perechi:
            d = d + p.getFrecventa()
        return d

    def iterator(self):
        return IteratorColectieFrecventa(self)


class IteratorColectieFrecventa:
    def __init__(self, c):
        self.__col = c
        self.__curent = 0
        self.__f = 1

    def valid(self):
        return self.__curent < len(self.__col._ColectieElementFrecventa__perechi)

    def element(self):
        return self.__col._ColectieElementFrecventa__perechi[self.__curent].getEl
ement()

    def urmator(self):
    #Distingem doua cazuri: frecventa curenta a elementului curent este (1) stric
t mai mica decat sau (2) egala cu frecventa elementului curent in colectia iterata

        # Daca frecventa curenta a elementului curent este strict mai mica decat
frecventa elementului curent in colectia iterata, incrementam frecventa curenta
        # Daca frecventa curenta a elementului curent este egala cu frecventa ele
mentului curent in colectia iterata, trecem la elementul urmator, incrementand ind
exul curent si reinitializand frecventa curenta cu 1
        if self.__f < self.__col._ColectieElementFrecventa__perechi[self.__curent]
.getFrecventa():
            self.__f = self.__f+1
        else:
            self.__curent = self.__curent+1
            self.__f=1

def populeazaColectieIntregi(c):
    c.adauga(1)
    c.adauga(2)
    c.adauga(3)
    c.adauga(2)
    c.sterge(2)
    c.adauga(2)

```

```
92. def tipareste(c):
93.     it = c.iterator()
94.     while it.valid():
95.         print(it.element())
96.         it.urmator()
97. 
98. def main():
99.     c = ColectieElementFrecventa()
100.     populeazaColectieIntregi(c)
101.     tipareste(c)
```