Nume: Buzilă Iulian-Ionel
Grupa: 212

Structuri de date și algoritmi

TAD Matrice – reprezentare folosind triplete <linie, coloană, valoare> (valoare ≠ 0); implementare sub forma unei liste dublu înlănțuite cu înlănțuirile reprezentate pe tablou.

Enunț

Într-un cartier rezidențial casele sunt așezate sub formă de matrice (matrice rară), unde elementele nenule sunt casele scoase la vânzare (valoarea reprezintă prețul), iar cele nule nu sunt de vânzare. Creați o aplicație care gestionează vânzări. Aplicația permite afișarea caselor oferite spre vânzare, adăugarea unei case spre vânzare, ștergerea unei case(vânzarea ei), modificarea prețului unei case).

Specificare și interfață:

- TAD Matrice Rară

MatriceRara = {**mr** | **mr** este o matrice rară cu elemente e = (l, c, v) - l, c de tip LCheie, v de tip TValoare.

creeaza(**mr, l, c**)
pre: -
post: mr ∈ **MatriceRara**, se creează o matrice rară vidă cu l linii si c coloane

adauga(**mr, l, c, v**)
pre: mr ∈ **MatriceRara**, l, c ∈ TCheie, v ∈ TValoare
post: mr' ∈ **MatriceRara**, mr' = mr + (l, c, v)
{s-a adăugat perechea (l, c, v)}

modifica(**mr, l, c, v**)
pre: mr ∈ **MatriceRara**, l, c ∈ TCheie, v ∈ TValoare
post: mr' ∈ **MatriceRara**, mr' = mr + (l, c, v)
{s-a modificat perechea (l, c, v)}

sterge(**mr, l, c**)
pre: mr ∈ **MatriceRara**, l, c ∈ TCheie
post: mr' ∈ **MatriceRara**, mr' = mr - (l, c, v)
{s-a șters perechea (l, c, v) }

nrLinii(**mr**)
pre: mr ∈ **MatriceRara**

post: l ∈ TCheie
{returnează numărul de linii ale matricei mr }

nrColoane(**mr**)
pre: mr ∈ **MatriceRara**
post: c ∈ TCheie
{returnează numărul de coloane ale matricei mr }

element(**mr, l, c**)
pre: mr ∈ **MatriceRara**, l, c ∈ TCheie
post: e ∈ TElement
{returnează elementul asociat cheilor l, c}

cauta(**mr, v**)
pre: mr ∈ **MatriceRara,** v ∈ TValoare
post: l, c ∈ TCheie
{returnează perechea (l, c) – linia și coloana unde a fost găsit elementul sau (0, 0) în caz contrar}

iterator(**mr, i**)
pre : mr ∈ **MatriceRara**
post : i ∈ **IteratorMatriceRara**; i este un iterator pe mr

- TAD Iterator matrice rară

IteratorMatriceRara ={ i | i este un iterator pe matrice rară}

creeaza(**i, mr**)
pre: mr ∈ MatriceRara
post: i ∈ **IteratorMatriceRara**
{se creează iterator pe mr. Elementul *curent* din iterator referă "primul" element din matricea rară}

element(**i, e**)
pre: i ∈ **IteratorMatriceRara**
post: e ∈ TElement, e este elementul *curent* din iterație (elementul din matricea rară referit de curent)

valid(**i**)
pre: i ∈ **IteratorMatriceRara**
post: valid = $\begin{cases} adevărat, \text{dacă iteratorul referă spre un element valid} \\ fals, \text{altfel} \end{cases}$
{funcția verifică dacă iteratorul referă spre un element valid din matricea rară}

urmator(**i**)

pre : i ∈ **IteratorMatriceRara,** *curent* este valid

post : *curent'* referă 'următorul' element din matricea rară față de cel referit de *curent*

precedent(**i**)

pre: i ∈ **IteratorMatriceRara,** *curent* este valid

post : *curent'* referă elementul 'precedent' din matricea rară față de cel referit de *curent*

prim(**i**)

pre: i ∈ **IteratorMatriceRara**

post: referă primul element al matricei rare iterată

{se setează iteratorul pe primul element din matricea rară}

Proiectare și implementare TAD

-folosind o listă dublu înlănțuită cu înlănțuirile reprezentate pe tablou

TElement

Linie: **Intreg**

Coloana: **Intreg**

Valoare: **Intreg**

Nod

element: ***TElement*** {elementul de tip ***TElement***}

următor: ***Intreg*** {poziția din vector a nodului următor}

anterior: ***Intreg*** {poziția din vector a nodului anterior}

MatriceRară

matrice: ***Nod[]*** {vector cu elementele din matrice, stocate in ***Nod***}

prim: ***Intreg*** {poziția primului element}

primLiber: ***Intreg*** {prima poziție liberă din vector}

ultim: ***Intreg*** {poziția ultimului element}

dim: ***Intreg*** {numărul elementelor din matrice}

IteratorMatriceRară

matrice: ***MatriceRară*** {matricea iterată}

curent: ***Intreg*** {poziția elementului curent din iterație}

Operații listă dublu înlănțuită cu înlănțuirile pe tablou

```
Subalgoritm creeaza(ListaD) este      θ(dimensiune)
        Pentru i←0, dimensiune exec
                ListaD[i].data ←TElement();
                ListaD[i].urmator←0;
                ListaD[i].precedent←0;
        Sfarsit pentru
        Prim←0
        primLiber←0
        ultim←0
sfarsit subalgoritm
```

```
Subalgoritm cautaPrimLiber(ListaD) este     θ(dimensiune)
        Pentru i←prim, dimensiune executa
                Daca ListaD[i].data = TElement(0,0,0) atunci
                        primLiber ← i
                        break;
                Sfarsit_daca
        Sfarsit_pentru
Sfarsit  subalgoritm
{Complexitate θ(dimensiune).
Caz favorabil O(1) – daca primLiber se afla pe prima pozitie atunci subalgoritmul se opreste
Caz defavorabil O(dimensiune) – daca primLiber se afla pe ultima pozitie din ListaD.
Caz mediu θ(primLiber) – daca primLiber se afla in interiorul listei
}
```

```
Subalgoritm adauga(ListaD, elem) este       θ(1)
        {temp – TNod}
        temp.data←elem
        temp.urmator ← 0;
        Daca prim = 0 atunci
                temp.precedent ← 0
                prim ← 1
                primLiber ← 2
                ListaD[1] ← temp
        Altfel
                ListaD[primLiber-1].urmator ← primLiber
                temp.precedent ← primLiber-1
                ListaD[primLiber] ←temp
                primLiber←primLiber+1
        sfarsit daca
sfarsit subalgoritm
```

```
subalgoritm adaugaDupa(ListaD, element, elNou) este      θ(pozitie)
        cat_timp ListaD[pozitie].data ≠ element exec
                pozitie ← ListaD[pozitie].urmator
        sfarsit_cat_timp
        cautaPrimLiber()
        temp.data ← elNou
        temp.precedent ← pozitie
        temp.urmator ← ListaD[pozitie].urmator
        ListaD[ListaD[pozitie].urmator].precedent ← primLiber
        ListaD[primLiber] ←  temp
        ListaD[pozitie].urmator ← primLiber
        ultim ← primLiber - 1
Sfarsit subalgoritm

Subalgoritm sterge(ListaD, element) este     θ(pozitie)
        pozitie←prim
        cat_timp ListaD[pozitie].data ≠ element exec
                pozitie←listaD[pozitie].urmator
        sfarsit_cat_timp
        daca pozitie = prim atunci
                prim ← prim + 1
                ListaD[ListaD[pozitie].urmator].precedent ← 0
                ListaD[pozitie].data ← TElement(0,0,0)
        Altfel
                Daca pozitie = ultim atunci
                        ListaD[ListaD[pozitie].precedent].urmator ← 0
                        ListaD[pozitie].data ← TElement(0,0,0)
                Sfarsit_daca
        Altfel
                ListaD[ListaD[pozitie].precedent].urmator ← ListaD[pozitie].urmator
                ListaD[ListaD[pozitie].urmator].precedent ← ListaD[pozitie].precedent
                Lista[pozitie].data ← TElement(0,0,0);
        Sfarsit_daca
        cautaPrimLiber()
Sfarsit  subalgoritm

Subalgoritm listaVida(ListaD) este    θ(dimensiune)
        vida←Adevarat
        pentru i←prim, dimensiune executa
                daca ListaD[i].data ≠ TElement(0,0,0) atunci
                        vida←Fals
                sfarsit_daca
        sfarsit_pentru
        return vida
Sfarsit  subalgoritm
```

Operații Iterator

```
Subalgoritm creeazaIterator(itr, mrr) este   θ(1)
        itr.curent ← m.lista.prim
        itr.m ← mrr
Sfarsit_subalgoritm
```

```
Subalgoritm urmator(itr) este          θ(1)
        Daca itr.curent < m.lista.dimensiune atunci
                itr.curent ← m.lista.ListaD[itr.curent].urmator
        Sfarsit_daca
Sfarsit_subalgoritm
```

```
Subalgoritm valid(itr) este     θ(1)
        Daca m.lista.ListaD[itr.curent].urmator ≠ 0 atunci
                Return Adevarat
        Sfarsit_daca
        Return Fals
Sfarsit_subalgoritm
```

Operații MatriceRară

```
Subalgoritm adaugaMatrice(lista, element) este     θ(1)
        Daca lista.listaVida() = Adevarat atunci
                lista.adauga(element)
        altfel
                elementAnterior←cauta(element)
                Lista.adaugaDupa(elementAnterior, element)
        sfarsit_daca
Sfarsit_subalgoritm
```

```
Subalgoritm getValoare(linie, coloana) este          θ(dimensiune)
        Pentru i←lista.prim, lista.dimensiune executa
                Daca lista.ListaD[i].data.getLinie() = linie si
                                lista.ListaD[i].data.getColoana() = coloana atunci
                        return lista.ListaD[i].data.getValoare();
                Sfarsit_daca
        Sfarsit_pentru
        Return 0
Sfarsit_subalgoritm
```

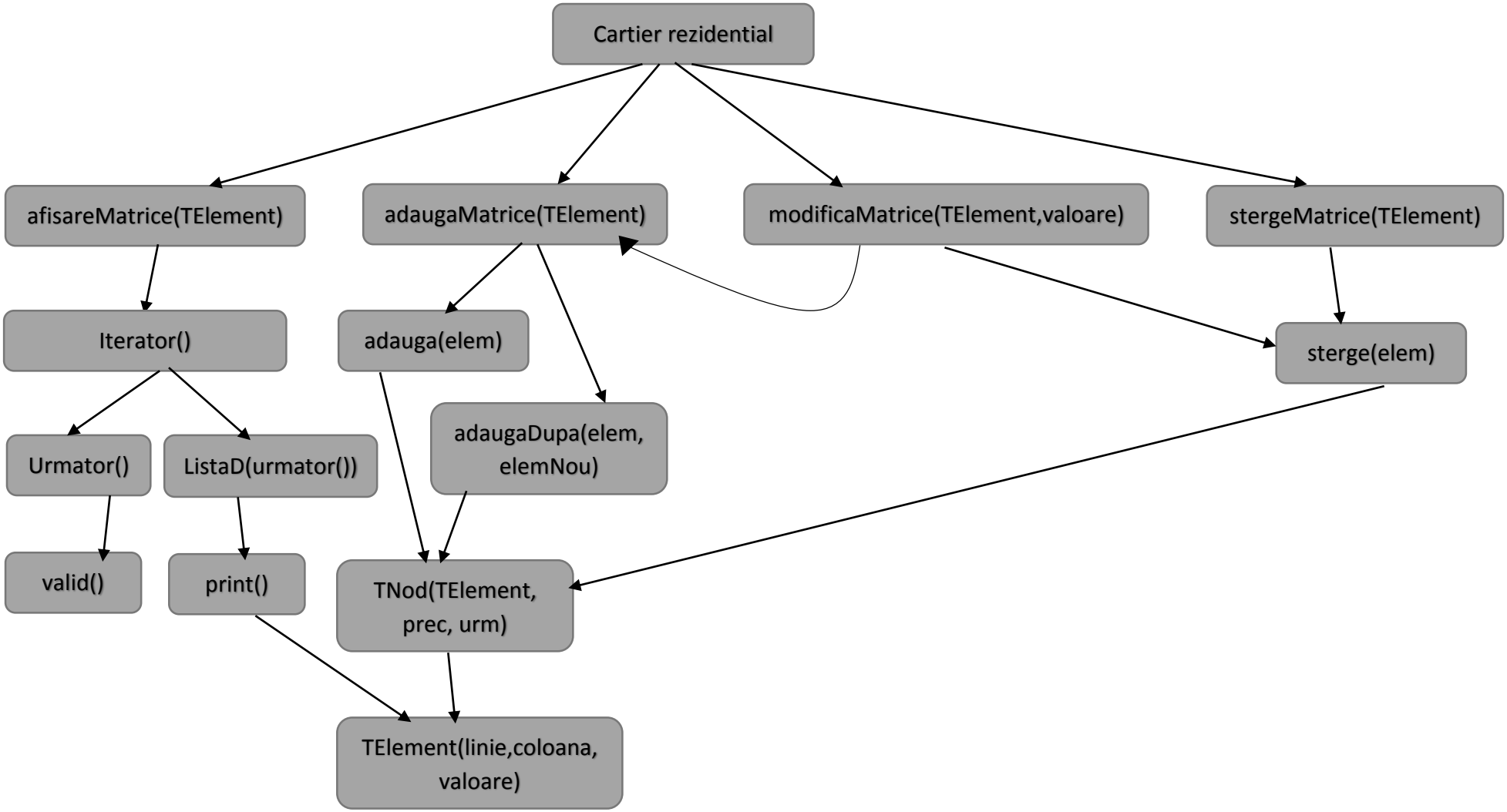
```
Subalgoritm cauta(lista, element) este        θ(dimensiune * dimensiune)
        linie ← element.getLinie()
        coloana ← element.getColoana()
        valoare ← element.getValoare()
        linieNoua ← 0
        coloanaNoua ← 0
        valoareNoua ← 0
        pentru linieV←0, lista.dimensiune executa
                pentru coloanaV←0, lista.dimensiune executa
                        daca coloanaV = coloana si linie = linie atunci
                                return TElement(linieNoua, coloanaNoua, valoareNoua)
                        altfel
                                daca lista.getValoare(linieV, coloanaV) ≠ 0 atunci
                                        linieNoua ← linieV
                                        coloanaNoua ← coloanaV
                                        valoareNoua ← lista.getValoare(linieV, coloanaV)
                                Sfarsit_daca
                        Sfarsit_daca
                Sfarsit_pentru
        Sfarsit_pentru
Sfarsit_subalgoritm

Subalgoritm Itr(lista){          θ(1]
        Return Iterator(lista)
Sfarsit_subalgoritm

Subalgoritm afisareMatrice(lista, iterator) este      θ(dimensiune)
        iterator←lista.Itr()
        lista.ListaD[iterator.curent].data.print();
        cat_timp iterator.valid() executa
                iterator.urmator()
                lista.ListaD[iterator.curent].data.print();
        Sfarsit_cat_timp
Sfarsit_subalgoritm

Subalgoritm stergeMatrice(lista, element) este      θ(1)
        Lista.sterge(element)
Sfarsit_subalgoritm
```

```
Subalgoritm modificareMatrice(lista, element, valoareNoua) este        θ(1)
        linie = element.getLinie()
        coloana = element.getColoana()
        valoare = element.getValoare()
        TElement vechi(linie, coloana, valoare)
        TElement nou(Linie, coloana, valoareNoua)
        lista.sterge(vechi)
        lista.adaugaMatrice(nou)
Sfarsit_subalgoritm
```


Cartier rezidential
afisareMatrice(TElement)
adaugaMatrice(TElement)
modificaMatrice(TElement,valoare)
stergeMatrice(TElement)
Iterator()
adauga(elem)
sterge(elem)
Urmator()
ListaD(urmator())
adaugaDupa(elem, elemNou)
valid()
print()
TNod(TElement, prec, urm)
TElement(linie,coloana, valoare)