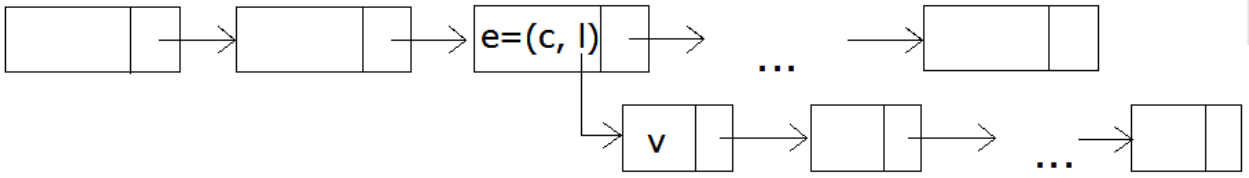
Multidicționar ordonat

- dicționar în care cheile sunt într-o anumită relație de ordine R (sunt memorate în ordine)
- o cheie are o listă de valori asociate
- o posibilitate de a reprezenta multidicționarul ordonat ar fi sub formă de listă, în care fiecare element este de fapt o pereche (cheie, valoare); deoarece pentru o cheie putem avea mai multe valori, cheile se pot repeta, însă ordinea cheilor se păstrează

Problema

Să se implementeze TAD Multidicționar ordonat – reprezentare înlănțuită cu alocare dinamică.



- un element *e* este reprezentat ca o pereche *(c, l)*, unde *c* reprezintă cheia, iar *l* este lista de valori asociată cheii respective
- fiecare valoare *v* a unei liste asociate unei chei aparține unui domeniu *TValoare*
- $c \in TCheie,\ R \subseteq TCheie \times TCheie$, *R* – este o relație de ordine pe chei

Reprezentare

Telement	*NodT*	*MultidictionarOrdonat*
c: TCheie	*e: TElement*	*prim: ↑NodT*
l: TLista	*urm: ↑NodT*	*R: relatie*

Operaţii

creează(mdo, R)
distruge(mdo)
adaugă(mdo, c, v) *// adaugă o pereche (c, v)*
şterge(mdo, c, v) *// șterge o pereche (c, v)*
caută(mdo, c, l) *// returnează lista de valori l asociate cheii c*
iterator(mdo, i) *// iterator pe multidicţionarul ordonat*
chei(mdo, m) *// mulţimea cheilor multidicţionarului*
valori(mdo, col) *// colecţia valorilor multidicţionarului*

Iterator

- 2 variante posibile:
  1. Reținem:
     - multidicționarul ordonat
     - referință spre nodul curent din multidicționarul ordonat
     - *Tpoziție* – poziția curentă din lista de valori asociată nodului curent

     Iterator:

     mdo: MultidictionarOrdonat
     curent: ↑NodT
     poz: TPozitie

  2. Reținem:
     - multidicționarul ordonat
     - referință spre nodul curent din multidicționarul ordonat
     - iterator pe lista valorilor asociată nodului curent

     Iterator:

     mdo: MultidictionarOrdonat
     curent: ↑NodT
     itL: IteratorLista

Pentru complexităţi, avem următoarele notaţii:

- n – numărul de chei distincte
- $|mdo|$ - numărul total de valori

Algoritmul de adăugare a unei perechi (cheie, valoare)

```
subalg adauga(mdo, c, v)                    {complexitate: O(|mdo|)}
        p ← cautNod(mdo, c)
        dacă p = NIL atunci
                adaugaCheieNoua (mdo, c, v)
        altfel
                adaugaSfarsit([p].e.l, v)   {operația de adăugare pe listă}
        sf-dacă
sf-subalg
```

```
subalg adaugaCheieNoua(mdo, c,v):           {complexitate: O(n)}
        aloca (q)                           {q va fi noul nod adaugat in lista}
        [q].e.c ← c
        creeazaL ([q].e.l)                  {operația de creare pe listă}
        adaugăSfarsit([q].e.l, v)           {operația de adaugare pe listă}
        daca mdo.prim = NIL atunci
                mdo.prim ← q
        altfel
                p ← mdo.prim
                dacă mdo.R(c, [p].e.c) atunci   {inseram inainte de primul nod}
                        [q].urm ← p
                        mdo.prim ← q
                altfel
                        cât-timp ([p].urm ≠ NIL si ¬ mdo.R(c, [[p].urm].e.c)) executa
                                p ← [p].urm
                        sf-cât-timp
                        {am gasit nodul p după care inseram}
                        [q].urm ← [p].urm
                        [p].urm ← q
                sf-dacă
        sf-dacă
sf-subalg
```

Algoritmul de ștergere a unei perechi (cheie, valoare)

```
subalg  șterge(mdo, c, v)                          {complexitate: O(|mdo|)}
        p ← cautNod(mdo, c)
        dacă p ≠ NIL atunci
                pos ← pozitieL([p].e.l, v)         {operația poziție, pe listă}
                dacă pos ≠⊥ atunci
                        ștergeL([p].e.l,  pos, e)  {operația de ștergere pe listă}
                sf-dacă
                dacă vidăL(([p].e.l) atunci        {operația vidă pe listă}
                        ștergCheie(mdo, c)
                sf-dacă
        sf-dacă
sf-subalg
```

```
subalg ștergCheie(mdo, c)                          {complexitate: O(n)}
        dacă [mdo.prim].e.c = c  atunci
                p ← mdo.prim
                mdo.prim ← [p].urm
                dealocă(p)
        altfel
                p ← mdo.prim
                cât-timp ([p].urm ≠ NIL) și ([[p].urm].e.c ≠ c) execută
                        p ← [p].urm
                sf-cât-timp
                {ștergem nodul de după p}
                q ← [p].urm
                [p].urm ← [q].urm
                dealocă(q)
        sf-dacă
sf-subalg
```