laborator 2

>> Structuri liniare I

Conţinut

- Structuri liniare în alocare statică: vectori
- Operaţii: traversare, inserare, ştergere
- Structuri liniare în alocare dinamică: liste înlănţuite
- Liste simplu înlănţuite
- Operaţii: traversare, căutare, inserare, ştergere

Referinţe

- **T.H. Cormen, C.E. Leiserson, R.L. Rivest.** *Introducere în algoritmi: cap 11.2 şi 11.3*, Editura Computer Libris Agora, 2000 (şi ediţiile ulterioare)
- **R. Ceterchi.** *Materiale de curs*, Anul universitar 2012-2013
- **http://laborator.wikispaces.com/**, *Tema 2*

O structură de date

Clasificare a structurilor liniare de date

- După tipul de alocare:
  - Structuri liniare în alocare statică (vectori)
  - Structuri liniare în alocare dinamică (liste înlănţuite)
- După modul de efectuare al operaţiilor de intrare (inserările) şi de ieşire (ştergerile):
  - Structuri liniare fără restricţii de intrare/ieşire
  - Structuri liniare cu restricţii de intrare/ieşire (stive şi cozi)

1. Structuri liniare în alocare statică (vectori)

Alocarea statică presupune că structura de date conţine elemente ce vor ocupa poziţii succesive în memorie. Acest fapt permite un acces foarte rapid la orice element din structură, deoarece nu este necesară traversarea altor elemente. În schimb, operaţiile de inserare şi de ştergere sunt costisitoare, pentru că se fac deplasări ale conţinutului elementelor.

A. Operaţia de traversare

Pentru parcurgerea unui vector `A` utilizăm un indice de traversare `k`. Acesta va fi iniţializat cu locaţia primului element din vector (pasul 1) şi va avansa la componenta următoare (pasul 4) cât timp mai există elemente de vizitat în vector. Acest test pentru nedepăşirea structurii este efectuat de pasul 2.

►► TRAVERSEAZĂ–VECTOR($A[1..n]$)

```
1.  k ← 1
2.  while k ⩽ n do
3.    Vizitează(A[k])
4.    k ← k+1
3.  endwhile
```

B. Operaţia de inserare

Următoarea procedură inserează valoarea `elem` pe poziţia `k` dintr-o structură liniară `A[1..n]`. În acest scop toate elementele de la poziţia `k` până la `n` sunt mutate câte o locaţie spre dreapta (paşii 1–5). Astfel, se eliberează poziţia `k` pentru a se putea realiza inserarea. Pasul 6 reprezintă inserarea propriu-zisă a valorii `elem`. După inserare dimensiunea vectorului `A` a crescut cu o unitate (pasul 7).

►► Inserează–În–Vector(A[1..n], k, elem)

```
1. i ← n
2. while i ⩾ k do
3.    A[i+1] ← A[i]
4.    i ← i-1
5. endwhile
6. A[k] ← elem
7. n ← n+1
```

C. Operaţia de ştergere

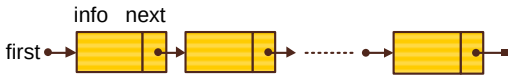
Procedura `Şterge-Din-Vector(A[1..n], k, x)` extrage în `x` valoarea `A[k]` (pasul 1) şi reface structura de vector a lui `A`. Acest lucru se realizează mutând toate elementele de la poziţia `k+1` până la `n` câte o locaţie la stânga (paşii 2–4) şi decrementând cu o unitate dimensiunea structurii (pasul 5), deoarece în urma ştergerii vectorul conţine cu un element mai puţin.

►► Şterge–Din–Vector(A[1..n], k, x)

```
1. x ← A[k]
2. for i ← k to n-1 do
3.    A[i] ← A[i+1]
4. endfor
5. n ← n-1
```

2. Structuri liniare în alocare dinamică (liste înlănţuite): Liste simplu înlănţuite

Spre deosebire de structurile liniare în alocare statică (tablouri), în care indicii tabloului determină ordinea elementelor, într-o listă (simplu) înlănţuită ordinea este determinată de un pointer conţinut în fiecare obiect. Listele înlănţuite asigură o reprezentare mai flexibilă pentru mulţimi dinamice.



Listele simplu înlănţuite sunt o colecţie de noduri care conţin fiecare:

*Elementele listei se numesc **noduri**.*

1. un câmp `info` pe care se reprezintă un element al mulţimii reprezentate prin listă (numit de obicei *câmp cheie*, deoarece elementul se identifică cu valoarea de pe un singur câmp); algoritmii discutaţi pornesc de la ipoteza că un element din mulţime ocupă un singur câmp;
2. un pointer `next` (o adresă) către nodul următor.

Faptul că structura este liniară înseamnă că fiecare nod (în afară de ultimul) are un singur nod succesor şi fiecare nod în afară de primul) are un singur nod predecesor.

În C++ puteţi reţune un nod folosind structuri:

```
struct nod {
  int info;
  nod *next;
};
```

Declararea unui pointer către o structură de tipul `nod`:

```
nod *first, *second;
```

Alocarea de memorie folosind operatorul `new` pentru o variabilă de tip pointer către structura de tip `nod`:

```
first = new nod;
second = new nod;
```

Modificarea câmpurilor din structura `nod`:

```
first->info = 1;
first->next = second;
second->info = 2;
second->next = 0; // null
```

Eliberarea memoriei utilizând operatorul `delete` :

```
delete first;
delete second;
```

Operaţiile de inserare şi ştergere pe liste simplu înlănţuite sunt mai rapide, deoarece nu se mai fac deplasările de elemente, dar pentru a accesa un nod din listă trebuie parcurse toate nodurile ce îl preced. În plus, reţinerea unui câmp de adresă în fiecare nod ocupă memorie.

Recursiv, o listă `L` de un anumit tip de bază este:

(a) fie lista vidă ($L = \emptyset$)

(b) fie o listă nevidă, caz în care conţine un nod numit **capul listei**, urmat de o altă listă de acelaşi tip de bază ($L =$`first`$|L_1$).

*A se înţelege prin **tip de bază** tipul de date al câmpului `info`.*

Obs.: Listele simplu înlănţuite se pot reprezenta şi prin structuri în alocare statică (*reprezentarea cu cursori a listei*):

- câmpurile `info` se află la anumite locaţii ale unui vector, iar
- câmpurile `next` asociate vor conţine indicele la care se află elementul următor.

A. Operaţia de traversare

Structura de listă simplu înlănţuită nu se poate parcurge decât într-un singur sens. Întâi se accesează primul nod, `first`. Apoi, din fiecare nod curent, numit `curent`, se accesează nodul următor cu ajutorul adresei reţinute de `curent->next`.

Observaţi că indicele curent *din procedura `Traversează-Vector(`*A[1..n]*`)` pentru structuri alocate static este înlocuit de un pointer către nodul curent.*

La pasul 1, pointerul pentru traversare (`curent`) este iniţializat, preluând adresa primului element din listă. Apoi, cât timp mai sunt elemente în structură (mai există adrese de parcurs), se vizitează nodul curent şi se avansează către următorul nod (paşii 2-5). Testul de nedepăşire a structurii este realizat prin condiţia testată în ciclul **`while`** : un câmp `next` care este **`NIL`** indică sfârşitul listei.

►► Traversează–Listă–Simplu–Înlănţuită(first)

```
1.  curent ← first
2.  while curent ≠ NIL do
3.    Vizitează(curent)
4.    curent ← curent->next
5.  endwhile
```

B. Operaţia de căutare

Următoarul algoritm caută în lista indicată de `first` valoarea `val`. Dacă găseşte un nod, al cărui câmp `info` conţine `val`, atunci returnează în variabila `loc` adresa acestui nod. Pointerul `loc` are proprietatea că `loc->info=val`. Dacă un asemenea nod nu se găseşte, atunci `loc` va fi **`NIL`** .

►► Caută–În–Listă–Simplu–Înlănţuită(first, val, loc)

```
1.  loc ← first
2.  while (loc ≠ NIL ) and (loc->info ≠ val) do
3.    loc ← loc->next
5.  endwhile
6.  if loc ≠ NIL then
7.    Căutarea s-a terminat cu succes.
8.  else
9.    Căutarea s-a terminat fără succes.
10. endif
```

C. Operaţia de inserare

În momentul creării unui nod nou se alocă spaţiu în mod dinamic.
Următoarea secvenţă de instrucţiuni creează un nod nou care să conţină valoarea x:

```
nod *nou = new nod;
nou->info = x;
nou->next = 0; // null (NIL)
```

Întâi s-a realizat alocarea de memorie, apoi s-a iniţializat câmpul cheie al nodului şi la sfârşit s-a atribuit valoarea **NIL** câmpului de legătură al nodului.

Operaţia de inserare propriu-zisă într-o listă simplu înlănţuită este operaţia care conectează nodul nou creat de celelalte noduri din listă, într-un loc anume în listă, care trebuie determinat în funcţie de natura problemei. După ce s-a determinat locul inserării, legarea noului nod la listă se face prin realocarea a doi pointeri (sau prin schimbul a două legături).

Următoarea secvenţă de instrucţiuni inserează un nod `nou` în capul listei:

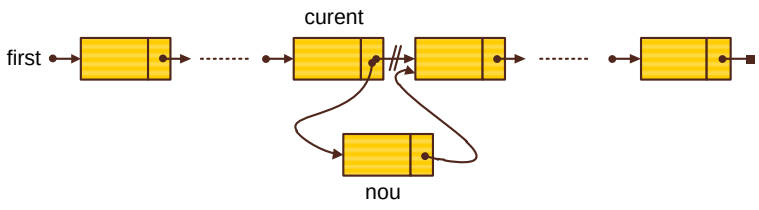
```
nou->next = first;
first = next;
```

Determinarea locului inserării în listă se face printr-o traversare eventual incompletă a listei cu un pointer `curent`. Traversarea va depinde de două condiţii:

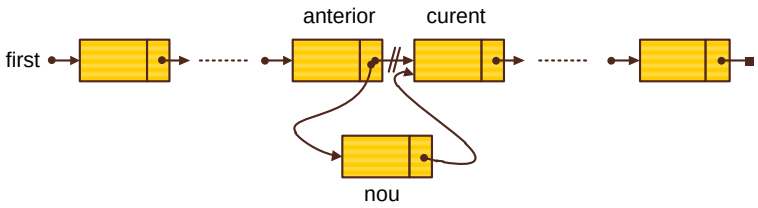
- o condiţie de nedepăşire a structurii, şi
- o condiţie referitoare la locul în care se face inserarea.

În funcţie de a doua condiţie, traversarea se poate termina într-una din următoarele situaţii:

1. pe un nod `curent` după care urmează să fie inserat nodul `nou` cu următoarea secvenţă de instrucţiuni:
   ```
   nou->next = curent->next;
   curent->next = nou;
   ```
   Acestea refac legăturile la stânga şi la dreapta.



2. pe un nod `curent` înaintea căruia urmează să fie inserat nodul `nou`. În acest caz, traversarea trebuie făcută cu doi pointeri succesivi: `anterior` şi `curent`.



Următoarea procedură inserează nodul `nou` înainte de primul nod `curent` pentru care avem `curent->info = val`. Paşii 3–6 efectuează o traversare incompletă a structurii. Inserarea între cei doi pointeri, `anterior` şi `curent` este realizată de paşii 10–15, după cum urmează:

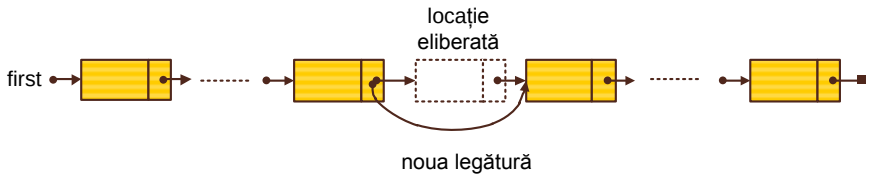
- legătura la stânga se face prin pasul 11, dacă inserăm în capul listei (adică nu există nod anterior, variabila `anterior` este **`NIL`**), sau prin pasul 13 dacă inserarea se face după `anterior`;
- legătura la dreapta se face prin pasul 15.

►► Inserează–În–Listă–Simplu–Înlănţuită(first, nou, val)

```
1.  anterior ← NIL
2.  curent ← first
3.  while curent ≠ NIL  and curent->info ≠ val do
4.    anterior ← curent
5.    curent ← curent->next
6.  endwhile
7.  if curent = NIL then
8.    Nu s-a găsit locaţia.
9.  else
10.   if anterior = NIL then
11.     first ← nou
12.   else
13.     anterior->next ← nou
14.   endif
15.   nou->next ← curent
16. endif
```

D. Operaţia de ştergere/extragere

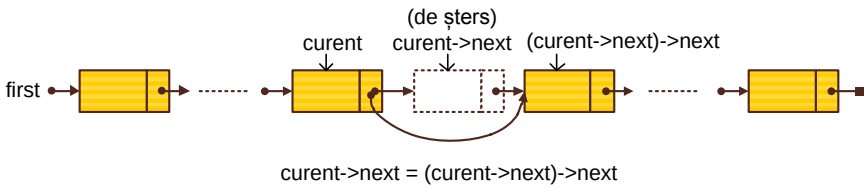
După efectuarea ştergerii sau extragerii nodului propriu-zise, operaţia de ştergere trebuie să prevadă şi refacerea structurii de listă simplu înlănţuită pe nodurile rămase. Pentru nodul extras se poate dealoca spaţiul sau efectua alte prelucrări.



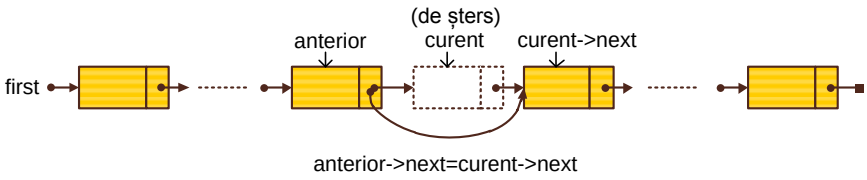
Similar inserării, poziţia nodului ce trebuie şters se găseşte în urma unei traversări eventual incomplete a listei, folosind un pointer `curent`. Traversarea este de asemenea guvernată de două condiţii: nedepăşirea structurii şi o condiţie specifică problemei. A doua condiţie conduce la terminarea traversării într-una din următoarele două situaţii:

A. pe un nod `curent` după care urmează nodul ce trebuie şters (se va şterge `curent->next`). Ştergerea se face prin instrucţiunea:

```
curent->next = (curent->next)->next;
```



B. pe un nod `curent` care este chiar nodul ce trebuie şters. În acest caz, ca şi la inserare, traversarea trebuie făcută cu doi pointeri succesivi: `anterior` şi `curent`.



Procedura următoare şterge dintr-o listă simplu înlănţuită primul nod care conţine valoarea `val`. Nodul şters va fi întâi extras într-un nod auxiliar `x`, ce nu face parte din listă.

►► ŞTERGE–DIN–LISTĂ–SIMPLU–ÎNLĂNŢUITĂ(first, val, x)

```
1.  curent ← first
2.  anterior ← NIL
3.  while (curent ≠ NIL and curent->info ≠ val) do
4.    anterior ← curent
5.    curent ← curent->next
6.  endwhile
7.  if curent ≠ NIL then
8.    x ← curent
9.    if anterior = NIL then
10.     first ← curent->next
11.   else
12.     anterior->next ← curent->next
13.   endif
14. endif
```

Dacă dorim să nu efectuăm o extragere ci doar o ştergere (nu avem nevoie de `x`), atunci la sfârşitul procedurii, mai exact după ce nu îl mai accesăm pe `curent` (pointer către acelaşi nod ca şi `x`) – vezi liniile 10 şi 12, putem să îl „distrugem" pe `x`, eliberând memoria alocată acestuia.

În C++ pentru eliberarea memoriei folosim operatorul `delete`

PROBLEME

1. **(1p)** Folosind algoritmul de traversare (cu prelucrare de chei) a unei structuri liniare alocate static A, de dimensiune n, cu elemente numere întregi, implementaţi:
   a. o funcţie `getMax` care primeşte ca parametrii pe n şi pe A şi returnează valoarea maximă din vector;
   b. o funcţie `getMinPos` care primeşte ca parametrii pe n şi pe A şi returnează poziţia din vector la care se găseşte elementul minim din structură.
2. **(1p)** Implementaţi algoritmul de inserare a unui element într-o structură liniară alocată static A, de dimensiune iniţială n, cu elemente numere întregi.
3. **(1p)** Implementaţi algoritmul de ştergere a unui element dintr-o structură liniară alocată static A, de dimensiune iniţială n, cu elemente numere întregi.
4. **(2p)** Implementaţi o funcţie `removeDuplicates`, care primeşte ca parametrii vectorul A *sortat* (cu elemente numere întregi) şi dimensiunea n şi elimină cheile duble din A prin deplasări de elemente.
5. **(3p)** Implementaţi o listă simplu înlănţuită şi următoarele funcţii:
   a. `insertElement` pentru adăugarea unui element nou;
   b. `printList` pentru afişarea listei;
   c. `findElement` pentru căutarea unui element;
   d. `removeElement` pentru ştergerea unui element;
   e. `empty` pentru ştergerea întregii liste.
6. **(1p)** Implementaţi următoarele funcţii pentru inserarea unui nod într-o listă alocată dinamic:
   a. `prepend` pentru adăugarea unui element nou la începutul listei;
   b. `insert` pentru adăugarea unui element nou în interiorul listei;
   c. `append` pentru adăugarea unui element nou la sfârşitul listei;
7. **(2p)** Reprezentarea numerelor mari cu ajutorul unei liste liniare simplu înlănţuite se face folosind următoarea schemă:

   Numărul întreg 82564 este reprezentat ca listă punînd fiecare cifră în câte un nod. Scrieţi un program în care se citesc două numere „mari" şi se construieşte o listă în care se va salva suma lor.
8. O reprezentare prin coeficienţi a unui polinom $P = \sum_0^n a_k X^k$ de grad n este un vector de coeficienţi $a = (a_0, a_1, \ldots, a_n)$. Fiind date două polinoame P de grad n şi Q de grad m, scrieţi un program care calculează:
   a. **(2p)** produsul lor.
   b. **(2p)** $P(x_0)$, adică evaluează polinomul P într-un punct dat x_0.
9. **(3p)** Să se implementeze cu ajutorul unei liste liniare, simplu înlănţuite şi alocate dinamic, un polinom de grad n. Fiecare nod se va considera că reţine gradul fiecărui monom, precum şi coeficientul său. Structura poate fi definită astfel :

```
struct pol {
  int gr, coef;
  pol *next;
};
```

   Scrieţi un program care creează două polinoame implementate prin liste şi calculează şi afişează suma lor.

■ TERMEN DE PREDARE: **Săptămâna 4 (22-26 octombrie) inclusiv.**

■ DETALII: **Studenţii pot obţine un maxim de 14 puncte. Problemele 5 şi 6 sunt obligatorii. Problemele 1-4, 7–9 sunt suplimentare.**