Programare Logică și funcțională
Autor Maria NUȚU
Versiune: 10.2023

Laborator 1 - Săptămâna 1

Recursivitate - Descriere matematică

Scopul acestui lab este să vă (re)familiarizați cu modul recursiv de a gândi rezolvarea problemelor de mate-info. Vom folosi această abordare în programarea cu Prolog (programare logică) și în Lisp (programare funcțională).

O soluție recursivă a unei probleme poate fi descrisă utilizând un „model recursiv matematic" sau „model recursiv". Aceasta este descrierea matematică soluției voastre.

Exemplu: Determinati **recursiv** suma cifrelor impare ale unui număr..
Soluție:
Putem determina ultima cifră a unui număr folosind operatorul modulo și împărțind numărul la 10 putem scăpa de ultima cifră.

- Dacă ultima cifră este pară, nu trebuie să o luăm în considerare, în acest caz pur și simplu determinați recursiv suma cifrelor pentru numărul fără ultima cifră.
- Dacă ultima cifră este impară, o vom adăuga la suma cifrelor calculate recursiv pentru restul numărului.
- Ne vom opri când numărul va deveni 0, în în acest caz rezultatul este 0.

Abordarea de mai sus poate fi descrisă folosind următorul model recursiv:

$$sumDigits(number) = \begin{cases} 0, & number = 0, \\ number \ \% \ 10 + sumDigits(number \ div \ 10), & number\%10 \text{ este impar} \\ sumDigits(number \ div \ 10), & \text{altfel} \end{cases}$$

Toate problemele din acest laborator (și cele mai multe probleme din celelalte laboratoare, de asemenea) necesită lucrul cu liste (sau seturi, dar un set este pur și simplu o listă cu elemente unice). În modelul recursiv, listele sunt reprezentate prin enumerarea elementelor:

l1, l2, l3, ..., ln

Când lucrăm cu liste, avem doar câteva operații disponibile și există multe lucruri pe care nu le putem face:

- Nu avem acces la lungimea listei. Dacă avem într-adevăr nevoie de lungimea unei liste va trebui să scriem o funcție **recursiva** care să o **calculeze**.
- Totuși, putem verifica dacă o listă conține un anumit număr de elemente. Astfel, putem verifica:
  - Dacă lista este goală: n=0
  - Dacă lista conține un singur element: n=1
  - Dacă lista conține doar 2 elemente: n=2

Deoarece putem accesa DOAR elementele de la ÎNCEPUTUL liste și doar un număr constant de elemente. Când folosim un anumit element de la începutul liste, avem acces și la restul listei în următoarea manieră:

- l1 - primul elem și l2,l3,...,ln - restul listei
  l2 - al doilea elem și l3,l4,...,ln restul listei

- NU putem compara lungimea unei liste cu un anumit număr (o constantă).
- NU putem accesa ULTIMUL element al listei.
- NU putem accesa un element de pe o anumită poziție (lk, k=constant)
- Nu putem adăuga elemente pe o anumită poziție k arbitrară (k=constant)
- Când soluția unei probleme este tot o listă, trebuie creată o listă noua în care vom adăuga elemente respective folosind operația de **reuniune**. Această listă nou creată va fi returnată ca soluție.
- NU putem modifica lista pe care am transmis-o ca parametru
- Putem **adăuga** elemente **doar** la **începutul** unei liste
- Nu putem concatena liste

Probleme

1. Calculați numărul de apariții al unui element într-o listă.
   Ex. occurrences ([1,2,3,4,5,2,4,2,2],2) = 4
   occurrences ([4,5,6,4,5],6) = 1

2. Fiind dată o listă, transformați lista astfel:
   a. Înmulțiți cu 5 numerele pare
   b. Scădeți 5 din numerele impare

3. Fiind dată o listă, adăugați după fiecare element de pe poziții pare suma cifrelor impare ale elementului respectiv. Primul element al listei se află pe poziția 1.
   Ex. AddAfterEvenPos([1,2,13,65,297,543,63]) = [1,2,**0**, 13, 65, **5**, 297, 543, **8,** 63]

4. Să se adauge un element la sfârșitul unei liste.
5. Să se șteargă aparițiile unui element dintr-o listă.