Seminar 1 - Recursivitatea

[marianzsu@cs.ubbcluj.ro](mailto:marianzsu@cs.ubbcluj.ro)

w[ww.cs.ubbcluj.ro/~marianzsu](http://Www.cs.ubbcluj.ro/~marianzsu)

Nota finala:

* 60% - examen scris
* 40% - nota laborator

Activitate Seminar => bonus (max 0.5p)

Prezenta: minim 5 din 7

* + Prezenti in call + sa aveti camera pornita + sa raspundeti daca sunteti intrebati

----------------------------------------------

1. Verificati daca un numar este norocos. Un numar este considerat norocos daca este alcatuit doar din cifrele 4 si 7. (de ex: 47, 747, 44, 777).

Norocos(n)=

* + - Fals, daca n%10 nu este 4 sau 7
    - True, n = 7 sau n = 4
    - norocos (n/10) daca n%10 este 4 sau 7

def Norocos(n):

if n==7 or n==4:

return True

if n%10!=4 and n%10!=7:

return False

return Norocos(n//10)

Python != minunat

1. Sa se calculeze suma divizorilor unui numar n. De ex: 20 => 2 + 4 + 5 + 10 => 21.

Sumdiv(divizor,n)= 0, daca divizor=1

divizor+ sumdiv(divizor-1,n), daca n%divizor

sumdiv(divizor-1,n), altfel

Apel\_sumdiv(n) = sumdiv(n/2,n) :)

Sumdiv(n,I) = 0, daca I\*I > n

I ,daca I\*I = n

I + n/i + Sumdiv(n,I+1) daca n%I==0

Sumdiv(n,I+1) altfel

Def sumdiv(d, n):

If d == 1:

Return 0

If n % d == 0:

Return d + sumdiv(d - 1, n)

Return sumdiv(d – 1, n)

Sumdiv(n//2,n)

**Liste:**

* O secventa de elemente in care fiecare element are o pozitie: l1l2...ln

|  |  |
| --- | --- |
| Se poate: | Nu se poate \* |
| * Putem compara lungimea cu o valoare CONSTANȚA   + N = 0 (e lista vida?)   + N = 1, n=3, n=10   + N < 2, n > 4, etc. * Putem accesa elemente de la inceputul listei (si numai un numar constant de elemente)   + L1 (primul element)   + L2, l3, l9 * Putem elimina elemente de la inceputul listei (si doar un numar constant de elemente)   + L2...ln   + L3...ln   + L5...ln   + L3...l10 * Putem adauga elemente doar la inceputul listei (si doar un numar constant de elemente)   + E U l1...ln   + E1 U E2 U l1...ln   + L1 U e U l2....ln | * Nu avem acces la lungimea listei   + Daca am 2 liste: l1l2...ln, m1m2...mk, nu putem compara lungimile   + Daca am o lista l1l2...ln si o pozitie p, nu putem verifica daca p e pozitie valida * Nu putem accesa orice element din lista   + Ln (ultimul element)   + Lp * Nu putem imparti lista la mijloc (la pozitii care nu sunt constante)   + L1...l(n-1)   + Lp...ln   + L1...lp * Nu putem adauga un element la sfarsitul listei si nu putem concatena 2 liste   + L1...ln U e   + M1...mk U l1...ln * Nu putem modifica o lista existenta   + L1 = e |

\*Se poate daca implmentam o functie recursiva pt operatia respectiva

Ex: sa se stearga elementul de pe pozitia p dintr-o lista

* returneaza lista nemodificata, daca p este mai mare decat lugimea listei

1. Sa se calculeze produsul numerelor pare dintr-o lista.

ProdusPare(l1l2l3...ln) = 1 daca n=0

l1 \* ProdusPare(l2l3l4...ln) daca l1%2==0

ProdusPare(l2l3l4...ln), altfel

Presupunem ca avem TAD Lista implementata cu urmatoarele operatii:

* EVida(lista) - returneaza True sau False
* PrimElem(lista) - returneaza primul element din lista
* Sublista(lista) - returneaza lista fara primul element
* CreazaVida() - creaza si returneaza o lista vida
* AdaugaInceput(elem, lista) - adauga elem la inceputul listei si returneaza lista rezultat

Def produsPare(lista):

If eVida(list):

Return 1

If primElem(lista)%2==0:

Return primElem(lista)\*produsPare(Sublista(lista))

Return produsPare(Sublista(lista))

1. a. Sa se adauge o valoare e pe pozitia m (m >= 1) intr-o lista (indexam lista de la pozitia 1). De ex: e = 11, m = 4, l = [1,2,3,4,5,6] => [1,2,3,11,4,5,6].

Adauga(l1l2...ln, m, el, I) = () daca n=0

el+l1+Adauga(l2l3...ln,m, el, I+1) daca m=I

L1+Adauga(l2l3...ln,m,el,I+1), altfel

Adauga([1,2,3,4,5,6], 4, 11, 1) = [1,2,3,11,4,5,6]

1 U Adauga([2,3,4,5,6], 4, 11, 2) = 1 U [2,3,11,4,5,6] =>[1,2,3,11,4,5,6]

2 U Adauga([3,4,5,6], 4, 11, 3) = 2 U [3, 11, 4, 5, 6] => [2,3,11,4,5,6]

3 U adauga([4,5,6], 4, 11, 4) = 3 U [11,4,5,6] => [3, 11, 4,5,6]

11 U 4 U adauga([5,6], 4, 11, 5) = 11 U 4 U [5, 6] => [11, 4, 5, 6]

5 U adauga([6], 4, 11, 6) = 5 U [6] => [5, 6]

6 U adauga([], 4, 11, 7) = 6 U = [6]

[]

Adauga([1,2,3], 6, 11, 1) = [1,2,3]

1 U adauga([2,3], 6, 11, 2) = 1 U [2,3] => [1,2,3]

2 U adauga([3], 6, 11, 3) = 2 U [3] => [2,3]

3 U adauga([], 6, 11, 4) = 3 U [ ] => [3]

[]

Def adauga(lista,m,el,I):

If eVida(lista):

Return creazaVida()

If m==i:

Return AdaugaInceput(el,AdaugaInceput(PrimElem(lista),Adauga(Sublista(lista),m,el,I+1)))

Return AdaugaInceput(PrimElem(lista),Adauga(Sublista(lista),m,el,I+1))