3. Evitarea apelului recursiv - LISP

```
(defun F(e)
    (cond
        ((numberp e) 0)
        (t ((lambda (x)
                (cond
                    ((> x a) (+ (car e) (F (cdr e))))
                    (+ x)
                )
            )
            (F (car e))
        )
    )
)
```

4. Să se determine calea de la rădăcină către un nod dat.

- o funcție care det. dacă un nod se află într-un subarbore
- o funcție care construiește drumul de la rădăcină la nodul dat, folosind MAPCAR

Modele matematice:

$$există(e_1 \dots e_m, e) = \begin{cases} \emptyset, & m = 0 \\ t, & e_1 = e \\ există(e_1, e) \text{ || } există(e_2 \dots e_m, e), & e_1 \neq e \\ există(e_2 \dots e_m, e), & altfel \end{cases}$$

$$drum(e, x) = \begin{cases} \phi, & e \text{ e atom și } e \neq x \\ (x), & e = x \\ e_1 \oplus \bigcup_{i=2}^{n} drum(e_i, x), & există(e, x) = t \\ \phi, & altfel \end{cases}$$

dacă x se află în subar. actual, atunci se adaugă nodul părinte la drum și se caută restul drumului în subarborele

```
; exista (l - lista, e - Element)
; returneaza true daca e se afla in lista oricat de adanc ar fi ea.
; in caz contrar

(defun exista(l e)
    (cond
        ((null l) nil)
        ((and (atom (car l)) (equal (car l) e)) t)
        ((listp (car l)) (or (exista (car l) e)
                             (exista (cdr l) e)))
        (t (exista (cdr l) e))
    )
)

; drum (l - lista (Atom, x - Element)
; returneaza o lista cu toate nodurile din drumul de la rad.
; la nodul x

(defun drum (l x)
    (cond
        ((and (atom l) (equal l x)) (list x))
        ((atom l) nil)
        ((exista l x) (append (list (car l))
                              (mapcan #'(lambda (y)
                                          (drum y x))
                                      (cdr l))))
    )
)
```

5. Determinarea nr-ului de subliste pt. care primul atom numeric este impar.

Modele matematice:

$$primul(e_1 \dots e_n) = \begin{cases} \emptyset, & n = 0 \\ e_1, & e_1 \in nr. \\ primul(e_1), & e_1 \text{ e listă și } primul(e_1) \neq \emptyset \\ primul(e_2 \dots e_n), & altfel \end{cases}$$

$$sublista(e) = \begin{cases} 0, & e \text{ e atom} \\ 1 + \sum_{i=1}^{m} sublista(e_i), & e \text{ e listă și } primul(e) \neq \emptyset, \\ & primul(e) \text{ e impar} \\ \sum_{i=1}^{m} sublista(e_i), & altfel \end{cases}$$

```
(defun primul (e)
    (cond
        ((null e) nil)
        ((numberp (car e)) (car e))
        ((and (listp (car e)) (not (null (primul (car e)))))
            (primul (car e)))
        (t (primul (cdr e)))
    )
)

(defun sublista (e)
    (cond
        ((atom e) 0)
```

```
((cond (not (null (primul l))) (addp (primul l)))
 (+ 1 (apply #'+ (mapcar #'sublista l))))
 (+ (apply #'+ (mapcar #'sublista l)))
 )
)
```

2. Să se genereze lista permutărilor având prop. că val. absolută a dif. dintre 2 valori consecutive din permutare este <= 3.

Modele matematice:

candidat $(e_1 \dots e_n)$ =

1. e_1, $n > 1$
2. candidat $(e_2 \dots e_n)$, $n > 1$

lungime $(e_1 \dots e_n)$ = $\begin{cases} 0, & n = 0 \\ 1 + \text{lungime}(e_2 \dots e_n), & \text{altfel} \end{cases}$

perm (e, n, eg, coe) = $\begin{cases} coe, & n = eg \\ \text{perm}(e, n, eg+1, e \oplus coe), & \text{dacă } |e - coe_i| \leq 3 \ (coe = coe_1 \dots coe_m) \text{ și } e \text{ nu există în } coe, \text{ unde } e = \text{candidat}(e) \end{cases}$

permutare (e) =

perm $(e, n, 1, (e))$, unde n = lungime (e)
e = candidat (e)

main (e) = $\bigcup$ permutare (e)

```
candidat([H|-], H).
candidat([-|T], E):-
    candidat(T, E).

lungime([], 0).
lungime([-|T], N):-
    lungime(T, N1),
    N is N1 + 1.

perm(L, N, N, Coe, Coe).
perm(L, N, Lg, [H|T], R):-
    candidat(L, E),
    not(candidat([H|T], E)),
    abs(E - H) =< 3,
    Lg is Lg + 1,
    perm(L, N, Lg1, [E|[H|T]], R).

permutare(L, R):-
    candidat(L, E),
    lungime(L, N),
    perm(L, N, 1, [E], R).

main(L, R):-
    findall(R1, permutare(L, R1), R).
```

(i, e) , (i, i) -> nedeterminist

(i, e) -> determinist

(i, i, i, i, e) ↓ nedeterminist

(i, e) -> nedeterminist

(i, e) -> determinist