

Recapitulare pentru Colocviu

În această secțiune se recapitulează conceptele de:

- Operații aritmetice
- Operații pe Liste (complete, diferență, incomplete)
- Liste Adânci
- Arbori (compleți, incompleți)
- Grafuri și efecte laterale (assert, retract)

1 Operații aritmetice

1. Calculați cel mai mare divizor comun a două numere.
?- cmmdc(15,25,R).
R = 5.
2. Calculați cel mai mic multiplu comun a două numere.
?- cmmmc(15,25,R).
R = 75.
3. Calculați divizorii unui număr natural.
?- divisor(15,R1), divisor(2,R2), divisor(1,R3), divisor(0,R4),divisor(6,R5).
R1 = [1,3,5,15], R2 = [1,2], R3 = [1], R4 = alot, R5 = [1,2,3,6].
4. Converteți un număr în binar (puterile lui 2 cresc de la dreapta la stânga)
?- to_binary(5,R1),to_binary(8,R2),to_binary(11,R3).
R1 = [1,0,1], R2 = [1,0,0,0], R3 = [1,0,1,1].
5. Inversați un număr natural.
?- reverse(15,R1), reverse(121235124,R2).
R1 = 51, R2 = 421542121.

2 Operații pe Liste

6. Calculați suma elementelor unei liste.
?- sum([1,2,3,4,5], R).
R = 15.
7. Dublați elementele impare și ridicați la pătrat cele pare.
?- numbers([2,5,3,1,1,5,4,2,6],R).
R = [4,10,6,2,2,10,16,4,36].
8. Extrageți numerele pare în E și numerele impare în O.
?- separate_parity([1,2,3,4,5,6], E, O).
E = [2,4,6], O=[1,3,5].
9. Înlocuiți toate aparițiile lui X cu Y.
?- replace_all(1, a, [1,2,3,1,2], R).
R = [a,2,3,a,2].
10. Înlocuiți toate aparițiile lui of X într-o listă diferență (al doilea si al treilea argument) cu secvența [Y,X,Y].
% replace_all(X, S, E, Y, R), where the difference list is S-E = [1,2,3,4,2,1,2]
?- replace_all(2,[1,2,3,4,2,1,2,2,3],[2,3],8,R).
R = [1,8,2,8,3,4,8,2,8,1,8,2,8].
11. Ștergeți aparițiile lui X pe poziții pare (numerotatea poziției începe de la 1).
?- delete_pos_even([1,2,3,4,2,3,3,2,5],2,R).
R = [1,3,4,2,3,3,5].
12. Ștergeți elementele de pe poziții divizibile cu K.
?- delete_kth([6,5,4,3,2,1], 3, R).
R = [6,5,3,2].
13. Ștergeți elementele de pe poziții divizibile cu K de la finalul listei.
?- delete_kth_end([1,2,3,4,5,6,7,8,9,10],3,R)
R = [1,3,4,6,7,9,10].
14. Ștergeți toate aparițiile elementului minim/maxim dintr-o listă.
?- delete_min([4,5,1,2], R).
R = [4,5,2].
15. Ștergeți elementele duplicate dintr-o listă (păstrează prima sau ultima apariție).
?- delete_duplicate([3,4,5,3,2,4], R).
R = [3,4,5,2]. sau R = [5,3,2,4].

16. Inversează o listă incompletă.
?- reverse([1, 2, 3, 4, 5 | _], R).
R = [5, 4, 3, 2, 1 | _].
17. Inversați elementele dintr-o listă după poziția K.
?- reverse_k([1,2,3,4,5,6], 2, R).
R = [1,2,6,5,4,3].
18. Codificați o listă cu RLE (Run-length encoding). Elemente consecutive se înlocuiesc cu (*element*, *nr_apariții*).
?- rle_encode([a,a,a,a,b,c,c,a,a,d,e,e,e,e], R).
R = [[a,4], [b,1], [c,2], [a,2], [d,1], [e,4]].
19. Codificați o listă cu RLE (Run-length encoding). Două sau mai multe elemente consecutive se înlocuiesc cu (*element*, *nr_apariții*). Dar dacă numărul de apariții este egal cu 1 atunci se scrie doar elementul.
?- rle_encode1([1,1,1,2,3,3,4,4], R).
R = [(1,3), 2, (3,2), (4,2)].
20. Decodificați o listă cu RLE (Run-length encoding).
?- rle_decode([[a,4], [b,1], [c,2], [a,2], [d,1], [e,4]], R).
R = [a,a,a,a,b,c,c,a,a,d,e,e,e,e].
21. Rotiți lista K poziții în dreapta.
?- rotate_k([1,2,3,4,5,6 | _], 2, R).
R = [5,6,1,2,3,4 | _].
22. Sortați o listă de caractere în funcție de codul ASCII.
?- sort_chars([e, t, a, v, f], R).
R = [a, e, f, t, v].
23. Sortați o listă de liste în funcție de lungimea listelor de nivel 2.
?- sort_len([[a, b, c], [f], [2, 3, 1, 2], [], [4, 4]], R).
R = [[], [f], [4, 4], [a, b, c], [2, 3, 1, 2]].
24. Stergeți elementele duplicate de pe poziții impare dintr-o listă (indecșii încep de la 1).
?- remove_dup_on_odd_pos([1,2,3,1,3,3,3,9,10,6,10,8,7,3], R).
R = [2,1,3,9,6,8,7,3].

3 Liste Adânci

25. Calculați adâncimea maximă a unei liste imbricate.

?- depth_list([1, [2, [3]], [4]], R1), depth_list([], R2).

R1 = 3, R2 = 1.

26. Aplatizați o listă imbricată cu liste complete/incomplete.

?- flatten([[1|_], 2, [3, [4, 5|_|_|_|_]], R).

R = [1,2,3,4,5|_].

27. Aplatizați doar elementele de la o adâncime dată într-o listă imbricată.

?- flatten_only_depth([[1,5,2,4],[1,[4,2],[5,[6,7,8]]],[4,[7]],8,[11]],3,R).

R = [4,2,5,7].

28. Calculați suma elementelor de la nivelul K într-o lista imbricată.

?- sum_k([1, [2, [3|_|_|_], [4|_|_|_]], 2, R).

R = 6.

29. Calculați numărul de liste într-o listă imbricată.

?- count_lists([[1,5,2,4],[1,[4,2],[5]],4,[7]],8,[11]],R).

R = 8.

30. Înlocuiți toate aparițiile lui X cu Y în lista imbricată.

?- replace_all_deep(2, 5, [[1, [2, [3, 2]], [4]], R).

R = [1, [5, [3, 5]], [4]].

31. Înlocuiți fiecare secvență cu o adâncime constantă cu lungimea într-o listă adâncă.

?- len_con_depth([[1,2,3],[2],[2,[2,3,1],5],3,1],R).

R = [[3],[1],[1,[3],1],2].

4 Arbori

32. Calculați adâncimea unui arbore binar complet/incomplet.

```
tree(t(6, t(4, t(2, nil, nil), t(5, nil, nil)), t(9, t(7, nil, nil), nil))).  
?- tree(T), depth_tree(T, R).  
R = 3.
```

33. Colectați toate nodurile unui arbore binar complet/incomplet în inordine folosind liste complete.

```
tree(t(6, t(4, t(2, nil, nil), t(5, nil, nil)), t(9, t(7, nil, nil), nil))).  
?- tree(T), inorder(T, R).  
R = [2,4,5,6,7,9].
```

34. Colectați toate frunzele dintr-un arbore binar.

```
tree(t(6, t(4, t(2, nil, nil), t(5, nil, nil)), t(9, t(7, nil, nil), nil))).  
?- tree(T), collect_k(T, R).  
R = [2,5,7].
```

35. Scrieți un predicat care verifică dacă un arbore este arbore binar de căutare.

```
tree(t(3, t(2, t(1, nil, nil), t(4, nil, nil)), t(5, nil, nil))).  
?- tree(T), is_bst(T).  
false.
```

36. Arbore binar incomplet. Colectați nodurile impare cu un singur copil într-o listă incompletă.

```
tree(t(26, t(14, t(2, _, _), t(15, _, _)), t(50, t(35, t(29, _, _), _), t(51, _, t(58, _, _)))).  
?- tree(X), collect_odd_from_1child(X, R).  
R = [35, 51 | _].
```

37. Arbore ternar incomplet. Colectați cheile între X și Y (interval închis) într-o listă diferență.

```
tree(t(2, t(8, _, _), t(3, _, t(4, _, _))), t(5, t(7, _, _), t(6, _, _), t(1, _, t(9, _, _)))).  
?- tree(T), collect_between(T, 2, 7, R, [1, 18]).  
R = [2, 3, 4, 5, 6, 7, 1, 18].
```

38. Arbore binar. Colectați cheile pare ale frunzelor într-o listă diferență.

```
tree(t(5, t(10, t(7, nil, nil), t(10, t(4, nil, nil), t(3, nil, t(2, nil, nil))))), t(16, nil, nil))).  
?- tree(T), collect_even_from_leaf(T, R, [1]).  
R = [4, 2, 16, 1].
```

39. Înlocuiți elementul minim dintr-un arbore ternar incomplet cu rădăcina.

```
tree(t(2,t(8,_,_),t(3,_,_t(1,_,_))),t(5,t(7,_,_),t(6,_,_),t(1,_,_t(9,_,_)))).  
?- tree(T), replace_min(T,R).  
R = t(2,t(8,_,_),t(3,_,_t(2,_,_))),t(5,t(7,_,_),t(6,_,_),t(2,_,_t(9,_,_)))).
```

40. Colectați toate nodurile de la adâncimea K dintr-un arbore binar.

```
tree(t(6, t(4, t(2, nil, nil), t(5, nil, nil)), t(9, t(7, nil, nil), nil))).  
?- tree(T), collect_k(T, 2, R).  
R = [4, 9].
```

41. Colectați toate nodurile de la adâncimi impare dintr-un arbore binar incomplet (rădăcina are adâncime 0).

```
tree(t(26,t(14,t(2,_,_),t(15,_,_)),t(50,t(35,t(29,_,_),_),t(51,_t(58,_,_)))).  
?- tree(X), collect_all_odd_depth(X,R).  
R = [14,50,29,58].
```

42. Colectați subarborii cu rădăcini conținând valoarea mediană dintr-un arbore ternar incomplet.

Observație. Mediana este "mijlocul" listei sortate de chei.

```
tree(t(2,t(8,_,_),t(3,_,_t(1,_,_))),t(5,t(7,_,_),t(5,_,_),t(1,_,_t(9,_,_)))).  
?- tree(T), median(T,R).  
R = [  
    t(5,t(7,_,_),t(5,_,_),t(1,_,_t(9,_,_))),  
    t(5,_,_)  
].
```

43. Înlocuiți fiecare nod cu înălțimea într-un arbore binar incomplet (frunzele au înălțimea 0).

```
tree(t(2,t(4,t(5,_,_),t(7,_,_)),t(3,t(0,t(4,_,_),_),t(8,_t(5,_,_)))).  
?- tree(T), height_each(T,R).  
R = tree(t(3,t(1,t(0,_,_),t(0,_,_)),t(2,t(1,t(0,_,_),_),t(1,_t(0,_,_)))).
```

44. Scrieți un predicat care înlocuiește întregul subarbore al unui nod (cu o cheie dată ca argument) cu un singur nod care are cheia suma cheilor subarborului acelui nod (dacă nu există un nod cu aceea cheie, rămâne neschimbat).

```
tree(t(14,t(6,t(4,nil,nil),t(12,t(10,nil,nil),nil)),t(17,t(16,nil,nil),t(20,nil,nil)))).  
?- tree(T), sum_subtree(T,6,R).  
R = t(14,t(32,nil,nil),t(17,t(16,nil,nil),t(20,nil,nil))).
```

5 Grafuri

45. Colectați toate nodurile unui graf.

```
node(1). node(2). node(3).  
?- collect(R).  
R = [1,2,3].
```

46. Calculați gradul interior/exterior al fiecărui nod dintr-un graf folosind predicatul dinamic *info(Node, OutDegree, InDegree)*.

```
edge(1,2). edge(2,1). edge(1,4). edge(1,3). edge(3,2).  
=> info(1,3,1). info(2,1,2). info(3,1,1). info(4,0,1).
```