



```

:- use_module(library(lists), [select/3]).

% draw(+G, -L): Az L folytonos vonal "megrajzolja" a G gráfot, azaz az
% L folytonos vonal ugyanazt a matematikai értelemben vett gráfot írja
% le, mint a G Prolog kifejezés.
draw(G, L) :-
    draw(G, _, L).

% draw(+G, ?KP, -L): Az L folytonos vonal a KP kezdőpontból indul és
% "megrajzolja" a G gráfot, azaz az L folytonos vonal ugyanazt a
% matematikai értelemben vett gráfot írja le, mint a G Prolog kifejezés.
draw([], _, []).
draw(G, P, [P-Q|L]) :-
    select_edge(P, Q, G, G1),
    draw(G1, Q, L).

% select_edge(P, Q, G, G1):
% A G irányítatlan gráfból elhagyható a P-Q él és marad a G1 gráf.
select_edge(P, Q, G, G1) :-
    select(E, G, G1),
    ( E = P-Q
    ; E = Q-P
    ).

% draw1(+G, -L): ugyanaz, mint draw/2, de csak egy
% segédeljárás használatával (select_edge/4)
draw1([], []).
draw1(G, [P-Q|L]) :-
    select_edge(P, Q, G, G1),
    ( G1 = [] -> L = []
    ; L = [Q-_|_],
      draw1(G1, L)
    ).

% draw2(+G, -L): ugyanaz, mint draw/2, csak segédeljárás nélkül.
draw2([], []).
draw2(G, [A-B|L]) :-
    select(E, G, G1),
    ( E = A-B
    ; E = B-A
    ),
    ( G1 = [] -> L = []
    ; L = [B-_|_],
      draw2(G1, L)
    ).

```

```

% 3. (Otthoni feladat)

% Írjon Prolog eljárást egy gráf fokszámlistájának előállítására. A
% fokszámlista típusa:

% :- type degrees == list(node_degree).
% :- type node_degree --> node - degree.
% :- type degree == int.

% A fokszámlista tehát egy olyan lista, amelynek elemei Cs-N alakú
% párok, ahol Cs a gráf egy csomópontja, és N a Cs csomópont
% fokszáma. A csomópontok tetszőleges sorrendben szerepelhetnek a
% fokszámlistában.

% degree_list(G, Ds): A G gráf fokszámlistája Ds.

% | ?- degree_list([b-a,a-c], Ds).
% Ds = [b-1,a-2,c-1] ? ; no

% degree_list(G, Ds): A G gráf fokszámlistája Ds.
degree_list([], []).
degree_list([A-B|G], Ds) :-
    degree_list(G, Ds0),
    incr_node_degree(Ds0, A, Ds1),
    incr_node_degree(Ds1, B, Ds).

% incr_node_degree(Ds0, A, Ds): A Ds0 fokszámlistából A fokszámának eggyel
% való növelésével áll elő a Ds fokszámlista.
incr_node_degree([], A, [A-1]).
incr_node_degree([N-D|Ds0], A, Ds) :-
    ( A = N -> D1 is D+1, Ds = [A-D1|Ds0]
    ; Ds = [N-D|Ds1],
      incr_node_degree(Ds0, A, Ds1)
    ).

% degree_list2(G, Ds): A G gráf fokszámlistája Ds. (Jobbrekurzív változat)
degree_list2(G, Ds) :-
    degree_list2(G, [], Ds).

% degree_list2(G, Ds0, Ds): A G gráf fokszámlistáját Ds0 elé
% fűzve kapjuk Ds-t.
degree_list2([], Ds0, Ds0).
degree_list2([A-B|G], Ds0, Ds) :-
    incr_node_degree(Ds0, A, Ds1),
    incr_node_degree(Ds1, B, Ds2),
    degree_list2(G, Ds2, Ds).

```

```

% 4. (Otthoni feladat)

% Írjon idraw/2 néven egy Prolog eljárást, amelynek jelentése azonos a
% 2. feladatban szereplő draw/2 eljárással! Törekedjék minél hatékonyabb
% megoldásra! Használhatja az ugraphs könyvtárat.

:- use_module(library(ugraphs)).

idraw(G, L) :-
    vertices_edges_to_ugraph([], G, Graph),
    symmetric_closure(Graph, SGraph),
    reduce(SGraph, [], % összefüggő a gráf
    degree_list2(G, Ds0),
    ( select(A-D1, Ds0, Ds1), D1 mod 2 == 1 ->
      select(B-D2, Ds1, Ds2), D2 mod 2 == 1,
      \+ ( member(_C-D3, Ds2), D3 mod 2 == 1 ),
      ( L = [A-_|_]
        ; L = [B-_|_]
        )
      )
    ; true
    ),
    draw(G, L).

% 5. Írjon Prolog eljárást amely a bemenetként kezdődő listáról eldönti,
% hogy egy platóval kezdődik-e, és ha igen, visszaadja a maximális plató
% hosszát és az ezutáni (maradék) elemek listáját.

% % pl_kezdetu(L, Len, M): Az atomokból álló L lista egy Len hosszúságú
% % maximális platóval kezdődik, amelyet az M maradéklista követ.

% | ?- pl_kezdetu([a,b,a,c,c,c,b,b], Len, M).
% no
% | ?- pl_kezdetu([c,c,c,b,b], Len, M).
% Len = 3, M = [b,b] ? ; no
% | ?- pl_kezdetu([b,b], Len, M).
% Len = 2, M = [] ? ; no
% | ?- pl_kezdetu([b], Len, M).
% no
% | ?- pl_kezdetu([], Len, M).
% no
% | ?-

% maxazonos(L0, M, A, N0, N): Az L0 lista elejéről leszedhető k db A és
% marad egy nem A-val kezdődő M, továbbá N=N0+k.
maxazonos(L0, M, A, N0, N) :-
    ( L0 = [A|L1] ->
      N1 is N0+1,
      maxazonos(L1, M, A, N1, N)
    ; M = L0, N = N0
    ).

pl_kezdetu([A,A|L], Len, M) :-
    maxazonos(L, M, A, 2, Len).

% Kevésbé hatékony, de segéd eljárás nélküli.
pl_kezdetu([A,A], 2, []).
pl_kezdetu([A|L], Len, M) :-
    L = [A|L1],
    L1 = [B|_],
    ( B = A -> pl_kezdetu(L, Len1, M), Len is Len1+1
    ; Len = 2, M = L1
    ).

```

```

% 6. Írjon olyan Prolog eljárást, amely felsorolja atomok egy adott
% listájában található maximális platókat, megadva a plató hosszát és az
% ismétlődő elemet.

% % plato(+L, ?Len, ?X): Az L listában található egy Len hosszúságú,
% % X elemekből képzett maximális plató.

% | ?- plato([a,b,b,b,b,a,a,c,b,b], Len, X).
% Len = 4, X = b ? ;
% Len = 2, X = a ? ;
% Len = 2, X = b ? ;
% no

plato(L, Len, X) :-
    L = [X0|L1],
    ( pl_kezdetu(L, Len0, M) ->
      ( X = X0, Len = Len0
        ; plato(M, Len, X)
        )
    ; plato(L1, Len, X)
    ).

:- use_module(library(lists), [last/2]).

plato2(L, Len, X) :-
    L2 = [X,X|_],
    append(L1, L23, L),
    append(L2, L3, L23),
    \+ L3 = [X|_],
    \+ last(L1, X),
    length(L2, Len).

% 7. (Otthoni feladat)

% Az előző feladat kiterjesztéseként írjon olyan Prolog eljárást, amely
% felsorolja atomok egy adott listájában található maximális platókat,
% megadva a plató kezdőindexét (1-től számozva), hosszát és az ismétlődő
% elemet.

% % plato(L, I, Len, X): Az L listában az I-edik elemtől kezdődően
% % egy X elemekből képzett, Len hosszúságú maximális plató található.

% | ?- plato([a,b,b,b,b,a,a,c,b,b], I, Len, X).
% I = 2, Len = 4, X = b ? ;
% I = 6, Len = 2, X = a ? ;
% I = 9, Len = 2, X = b ? ;
% no

plato(L, I, Len, X) :-
    plato(L, 1, I, Len, X).

plato(L, I0, I, Len, X) :-
    L = [X0|L1],
    ( pl_kezdetu(L, Len0, M) ->
      ( X = X0, Len = Len0, I = I0
        ; I1 is I0+Len0, plato(M, I1, I, Len, X)
        )
    ; I1 is I0+1, plato(L1, I1, I, Len, X)
    ).

```