```
%                     Deklaratív Programozás gyakorlat
%                    Prolog programozás: listák, gráfok
%                              MEGOLDÁSOK
% ------------------------------------------------------------


% 1.  Beszúrás rendezett listába

%     % insert_ord(+RL0, +Elem, RL): Az RL monoton növő számlista úgy áll
%     % elő, hogy az RL0 szigorúan növő számlistába beszúrjuk az Elem számot,
%     % feltéve hogy Elem nem eleme az RL0 listának; egyébként RL = RL0.

%     % insert_ord(+RL0, +Elem, ?RL): Az RL szigorúan monoton növő számlista
%     % úgy áll elő, hogy az RL0 szigorúan növő számlistába beszúrjuk az Elem
%     % számot, feltéve hogy Elem nem eleme az RL0 listának; egyébként RL = RL0.

%     | ?- insert_ord([1,3,5,8], 6, L).
%     L = [1,3,5,6,8] ? ;
%     no
%     | ?- insert_ord([1,3,5,8], 3, L).
%     L = [1,3,5,8] ? ;
%     no

%     Megoldása legyen jobbrekurzív, használjon feltételes szerkezetet!

% insert_ord(+RL0, +Elem, ?RL): Az RL szigorúan monoton növő számlista
% úgy áll elő, hogy az RL0 szigorúan növő számlistába beszúrjuk az Elem
% számot, feltéve hogy Elem nem eleme az RL0 listának; egyébként RL = RL0.
insert_ord([], E, [E]).
insert_ord(L0, E, L) :-
        L0 = [X|L1],
        (   X > E -> L = [E|L0]
        ;   X =:= E -> L = L0
        ;   L = [X|L2],
            insert_ord(L1, E, L2)
        ).

% insert_ord_1(+RL0, +Elem, ?RL): ugyanaz mint
%   insert_ord(+RL0, +Elem, ?RL), de feltételes szerkezet használata
% nélkül. Kevésbé hatékony, mert választási pontot hagy maga után.
insert_ord_1([], E, [E]).
insert_ord_1([E|L0], E, [E|L0]).
insert_ord_1([X|L0], E, [E,X|L0]) :-
        X > E.
insert_ord_1([X|L0], E, [X|L]) :-
        X < E,
        insert_ord_1(L0, E, L).


% insert_ord_2(+RL0, +Elem, ?RL): ugyanaz mint
%   insert_ord(+RL0, +Elem, ?RL), a vágó beépített eljárás használatával.
% Kicsit egyszerűbb mint az előző két változat, mert csak
% háromfelé ágazik (insert_ord_1 1. és 3. klózát itt most
% egyetlen klóz, a 3. helyettesíti)
insert_ord_2([X|L0], E, L) :-
        X < E, !, L = [X|L1], insert_ord_2(L0, E, L1).
insert_ord_2([E|L0], E, L) :-
        !, L = [E|L0].
insert_ord_2(L0, E, [E|L0]).


% ------------------------------------------------------------------------
```

```
% A 'graph' adatstruktúrát a következő Mercury-szerű típusdefiníciókkal
% definiáljuk:

% % :- type graph == list(edge).
% % :- type edge ---> node-node.
% % :- type node == atom.

% Eszerint egy Prolog kifejezés a 'graph' típusba tartozik, ha X-Y alakú
% struktúrák listája, ahol X és Y névkonstansok (atomok).

% Az [a1-b1,a2-b2,...,an-bn] 'graph' típusú kifejezés azt az irányítatlan
% gráfot írja le, amelynek csomópontjai  a1,..,an,b1,...,bn, és
% egy (irányítatlan) él vezet ai és bi között, minden i=1,...,n esetén.
% (Megjegyzés: az így megadott gráfoknak nyilván nem lehet izolált pontja.)



% 2a. Írjon egy graph/1 Prolog eljárást amely egy tetszőleges Prolog
%     kifejezésről eldönti, hogy az a 'graph' típusba tartozik-e.

%     % graph(G): G egy 'graph' típusba tartozó Prolog kifejezés.

%     | ?- graph([]).
%     yes
%     | ?- graph([a-b,b-c]).
%     yes
%     | ?- graph([a-b,b-1]).
%     no

%     Megoldása legyen jobbrekurzív!

graph([]).
graph([N1-N2|Es]) :-
        atom(N1), atom(N2), graph(Es).

% 2b. Írjon egy same_edge/2 Prolog eljárást amely két 'edge' típusú Prolog
%     kifejezésről eldönti, hogy azonos irányítatlan élet írnak-e le!

%     | ?- same_edge(a-b, a-b).
%     yes
%     | ?- same_edge(a-b, b-a).
%     yes
%     | ?- same_edge(a-b, b-b).
%     no

%     Az alábbi célok futtatásával vizsgálja meg, hogy az eljárás a (+,+)
%     módtól különböző módokban is működik-e.

%     | ?- same_edge(a-b, E).

%     | ?- same_edge(E, a-b).

%     | ?- same_edge(E1, E2).

same_edge(N1-N2, N1-N2).
same_edge(N1-N2, N2-N1).

% 2c. Írjon egy same_graph0/2 Prolog jobbrekurzív eljárást amely két 'graph'
%     típusú Prolog kifejezésről eldönti, hogy (matematikai értelemben)
%     azonos gráfot írnak-e le!

%     % same_graph0(G1, G2): G1 és G2 azonos gráfot írnak le.

%     | ?- same_graph0([], []).
%     yes
%     | ?- same_graph0([a-b], [a-b]).
%     yes
%     | ?- same_graph0([a-b], [b-a]).
%     yes
```

```
%     | ?- same_graph0([a-b,b-c], [b-c,b-a]).
%     yes
%     | ?- same_graph0([a-b,a-b], [a-b]).
%     no

%     Megvalósítási ötletek:

%     - Használja a same_edge/2 segédeljárást.
%     - Használja a select/3 eljárást, amely elérhető a lists könyvtárban,
%       illetve a 248. dián.

%     Vizsgálja meg, hogy helyesen és végesen működik-e az eljárás, ha csak
%     az egyik argumentumot adjuk meg (és a másik  változó). Például futtassa
%     az alábbi két eljáráshívást:

%     | ?- same_graph0([a-b], G).

%     | ?- same_graph0(G, [a-b]).

%     Ezek közül az egyik az első megoldás után végtelen választási pontot
%     hoz létre. A trace eljárás meghívásával kapcsolja be a nyomkövetést,
%     futtassa a megfelelő eljáráshívást és értse meg, hogy miért jön létre
%     végtelen választási pont. (Lásd még a 248. dia utolsó mondatát).

%     Becsülje meg az alábbi célsorozat megoldásainak számát:

%     | ?- house(G0), same_graph0(G, G0).

%     Az alábbi célsorozat futtatásával megállapíthatja a pontos
%     megoldásszámot:

%     | ?- house(G0), findall(1, same_graph0(G, G0), Gs), length(Gs, N).

:- use_module(library(lists), [select/3, same_length/2]).

same_graph0([], []).
same_graph0([E|FGT], SG) :-
        select(E1, SG, SGR),
        same_edge(E, E1),
        same_graph0(FGT, SGR).

% 2d. Olvassa el a same_length/2 lists könyvtárbeli eljárás specifikációját,
%     az SWI Prolog esetében pl. itt:
%     https://www.swi-prolog.org/pldoc/doc_for?object=same_length/2 (A kék
%     fejléc jobb szélén levő narancssárga körbe foglalt :- jelre kattintva
%     megnézheti az eljárás -- végtelenül egyszerű -- Prolog kódját is.)

%     A same_length/2 segítségével írjon egy same_graph(G0, G) eljárást,
%     amelynek jelentése ugyanaz, mint a same_graph0 eljárásé, de nem
%     érzékeny az argumentumok sorrendjére: ha legalább az egyik argumentuma
%     zárt végű lista, akkor véges a keresési tere.

%     Megvalósítási segítség: a same_graph egyetlen klózzal megvalósítható
%     úgy, hogy a klóz törzse csak két hívásból áll.


same_graph(FG, SG) :-
        same_length(FG, SG),
        same_graph0(FG, SG).

% 2e. Írjon egy line/2 eljárást, amely eldönti, hogy egy adott gráf egy adott
%     pontból kiinduló folytonos vonal-e.

%     % line(G, P):  A G gráf a P pontból kiinduló folytonos vonal.

%     | ?- line([], a).
%     yes
%     | ?- line([a-b], a).
%     yes
%     | ?- line([a-b,c-b], a).
```

```
%     no
%     | ?- line([a-b,b-c], a).
%     yes
%     | ?- line([a-b,b-c], c).
%     no

%     Vizsgálja meg, hogy a line eljárás hogy viselkedik, ha az első, G
%     argumentum változó, illetve ha változókból álló (zárt végű) lista.
%     Ehhez futtassa az alábbi célokat:

%     | ?- line(L, a).

%     | ?- length(L, 5), line(L, P).

%     Vegye észre, hogy ha L zárt végű, változókból álló lista (mint az
%     utolsó példában), akkor a line(L, P) hívás létrehoz egy olyan folytonos
%     vonalat, amelynek pontjai még változók (hála a Prolog "logikai"
%     változó-fogalmának, azaz annak, hogy a változók az adatstruktúrák
%     részei lehetnek).                                                    (*)

line([], _).
line([P-Q|L], P) :-
        line(L, Q).

%     A jelen feladat hátralevő alpontjaiban  az alábbi fejkommentnek
%     megfelelő draw/2 Prolog eljárás különböző változatait készítjük el.
%
%     % 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.
%
%     Tehát a "| ?- draw(G, L)." hívás, ahol G adott és L egy változó,
%     felsorolja L-ben az összes olyan folytonos vonalat, amely "megrajzolja"
%     G-t.
%
%     | ?- draw([a-b,a-c], L).
%     L = [b-a,a-c] ? ;
%     L = [c-a,a-b] ? ;
%     no
%     | ?- draw([a-b,a-c,b-c,b-d,b-e,c-d,c-e,d-e], L), L = [d-e|_].
%     L = [d-e,e-b,b-a,a-c,c-b,b-d,d-c,c-e] ? ;
%     L = [d-e,e-b,b-a,a-c,c-d,d-b,b-c,c-e] ? ;
%     L = [d-e,e-b,b-c,c-a,a-b,b-d,d-c,c-e] ? ;
%     L = [d-e,e-b,b-c,c-d,d-b,b-a,a-c,c-e] ? ;
%     L = [d-e,e-b,b-d,d-c,c-a,a-b,b-c,c-e] ? ;
%     L = [d-e,e-b,b-d,d-c,c-b,b-a,a-c,c-e] ? ;
%     L = [d-e,e-c,c-a,a-b,b-c,c-d,d-b,b-e] ? ;
%     L = [d-e,e-c,c-a,a-b,b-d,d-c,c-b,b-e] ? ;
%     L = [d-e,e-c,c-b,b-a,a-c,c-d,d-b,b-e] ? ;
%     L = [d-e,e-c,c-b,b-d,d-c,c-a,a-b,b-e] ? ;
%     L = [d-e,e-c,c-d,d-b,b-a,a-c,c-b,b-e] ? ;
%     L = [d-e,e-c,c-d,d-b,b-c,c-a,a-b,b-e] ? ;
%     no

house([a-b,a-c,b-c,b-d,b-e,c-d,c-e,d-e]).

% 2f. Írja meg a draw0(G, L) eljárást "generál és ellenőriz" (generate and test)
%     formában: generálja le a G gráffal matematikailag azonos L gráfokat,
%     majd válassza ki ezek közül azokat, amelyek folytonos vonalat alkotnak!

%     Megvalósítási ötlet:

%     A same_graph0/2 eljárás segítségével sorolja fel az adott G-vel azonos
%     L gráfokat (vigyázzon G és L megfelelő sorrendjére, vagy használja
%     same_graph/2 eljárást), majd line/2 segítségével válassza ki azokat
%     amelyek folytonos vonalak.
```

```
% draw0(+G, -L): Az L folytonos vonal "megrajzolja" a  G gráfot.
draw0(G, L) :-
        same_graph0(L, G),
        line(L, _).

% 2g. Írja meg a draw1(G, L) eljárást "ellenőriz és generál" (test and generate)
%     formában!

%     Megvalósítási ötlet: a 2e. variáns (*)-gal jelzett utolsó bekezdése
%     szerint a line(L, _) eljárás véges lefutásához elegendő, hogy L hossza
%     ismert legyen. Ennek biztosítására használja a same_length/2 eljárást,
%     majd a draw0/2 törzsében levő két hívást fordított sorrendben hajtsa
%     végre.

%     A draw1/2 eljárás kb 7500-szer gyorsabb mint a draw0/2!

% draw1(+G, -L): Az L folytonos vonal "megrajzolja" a  G gráfot.
draw1(G, L) :-
        same_length(G, L),
        line(L, _),
        same_graph(L, G).

% 2h. Írja meg a draw2/3 segédeljárást az alábbi fejkommentnek megfelelően!

%     % draw2(G, P, L): A G gráfot megrajzolja a P pontból kiinduló L
%     % folytonos vonal.

%     Megvalósítási ötlet: használja a select/3 és a same_edge/2 eljárásokat.

%     Az eljárás írásakor feltételezheti, hogy P ismert, de szinte biztos,
%     hogy akkor is működik az eljárás, ha P nincs behelyettesítve.

%     Az utóbbi észrevételre építve írja meg a draw/2 eljárás újabb
%     változatát draw2/2 néven, amely a draw2/3-ra vezeti vissza feladatot.

%     A draw2/2 eljárás lényegében azonos sebességű a draw1/2-vel.

% draw2(+G, -L): Az L folytonos vonal "megrajzolja" a  G gráfot.
draw2(G, L) :-
        draw2(G, _, L).

% draw2(+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.
draw2([], _, []).
draw2(G, P, [P-Q|L]) :-
        select(E, G, G1),
        same_edge(E, P-Q),
        draw2(G1, Q, L).

% 2i. Írja meg a draw3(G, L) eljárást úgy, hogy csak a select/3 könyvtári
%     eljárást használja, más segédeljárást nem.
%     A draw3/2 eljárás kb 5-10%-kal lassabb mint a draw1/2 és draw2/2.

% draw3(+G, -L): ugyanaz, mint draw2/2, csak segédeljárás nélkül.
draw3([], []).
draw3(G, [P-Q|L]) :-
        select(E, G, G1),
        (   E = P-Q
        ;   E = Q-P
        ),
        (   G1 = [] -> L = []
        ;   L = [Q-_|_],
            draw3(G1, L)
        ).
```

```
% draw_variant(Variant, Rep): Variant egy (kétargumentumú) gráfrajzoló
% eljárás neve, Rep az ismételt futtatások száma.
draw_variant(draw0, 1).
draw_variant(D, 10000) :-
        member(K, ['1','2','3']),
        atom_concat(draw, K, D).

% Időméréssel lefuttatja a draw variánsokat.
test_draw :-
        draw_variant(Pred, N),
        house(G0),
        Goal =.. [Pred,G0,_],
        statistics(runtime, _),
        (   between(1, N, _), Goal, fail
        ;   true
        ),
        statistics(runtime, [_,T0]),
        T is T0*1000//N,
        format('Pred:~|~t~w~6+, time to find all sols:~|~t~6d~11+ sec\n',
               [Pred, T]),
        fail.
test_draw.

% 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, amelyenek 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).



% Nagyméretű példagráf
big_house([a-b,a-c,b-c,b-d,b-e,c-d,c-e,d-e,
           a-f,f-b,f-c,g-f,g-b,g-c,h-g,h-b,h-c,
           i-h,i-b,i-c,j-i,j-b,j-c,k-j,k-b,k-c]).

%               a  ------ f ------ g ------ h ------ i ------ j ------ k
%              / \       / \      / \      / \      / \      / \      / \
%             /   \     +   +    +   +    +   +    +   +    +   +    +   +
%            /     \   b     c  b     c  b     c  b     c  b     c  b     c
%           b-------c
%           |\     /|
%           |  \ /  |
%           |  / \  |
%           |/     \|
%           d-------e

:- prolog_flag(dialect, sicstus), use_module(library(between), [between/3]) ; true.


% A nagyméretű gráfra időméréssel lefuttatja a Pred nevű variánst.
test_idraw(Pred) :-
    big_house(G0),
    length(G0, Len),
    N is (Len-8)//3,
    (   between(1, N, I),
        NoOfEdges is 8+3*I,
        length(G0, NoOfEdges),
        append(G0, _, G),
        Goal =.. [Pred,G,_],
        statistics(runtime, _),
        \+ Goal,            % Goal is bound to fail
        statistics(runtime, [_,T]),
        format('Pred:~|~t~w~6+, edges:~|~t~w~4+, time to fail:~|~t~3d~8+ sec\n',
               [Pred, NoOfEdges, T]),
        fail
     ;   true
     ).

% A nagyméretű gráfra időméréssel lefuttat két variánst.
test_idraw :-
    test_idraw(idraw), nl, test_idraw(draw2).
```

```
/*
| ?- test_idraw.
Pred: idraw, edges:  11, time to fail:   0.000 sec
Pred: idraw, edges:  14, time to fail:   0.000 sec
Pred: idraw, edges:  17, time to fail:   0.000 sec
Pred: idraw, edges:  20, time to fail:   0.000 sec
Pred: idraw, edges:  23, time to fail:   0.000 sec
Pred: idraw, edges:  26, time to fail:   0.000 sec
Pred:  draw, edges:  11, time to fail:   0.000 sec
Pred:  draw, edges:  14, time to fail:   0.050 sec
Pred:  draw, edges:  17, time to fail:   1.120 sec
Pred:  draw, edges:  20, time to fail:  26.670 sec
Pred:  draw, edges:  23, time to fail: 699.370 sec
  C-c C-cProlog interruption (h for help)? a
% Execution aborted
| ?-
*/
% ------------------------------------------------------------------------
```

```
% Platónak hívunk egy olyan legalább kételemű listát, amely csupa azonos
% elemből áll. Az mondjuk, hogy MP egy L listában található maximális
% plató, ha
% - MP az L folytonos része (azaz MP előáll úgy, hogy L elejéről és
%   végéről 0 vagy több elemet elhagyunk),
% - MP egy plató és
% - MP maximális L-ben, azaz nem lehet sem balra sem jobbra a benne
%   levőkkel azonos, közvetlenül szomszédos elemekkel kiterjeszteni.

% Például az L = [a.b,a,c,c,c,b,b] listában két maximális plató van, a
% 4. pozición kezdődő [c,c,c] és a 7. pozición kezdődő [b,b] lista (a
% listaelemeket 1-től számozzuk).

% Egy adott listában előforduló platók felsorolására használható az
% alábbi deklaratív, de nem hatékony (potenciálisan négyzetes költségű)
% Prolog eljárás:

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

    plato0(L, I, Len, X) :-     % Az L listában az I. pozición van egy
                                % Len hosszúságú, X elemekből képzett
                                % maximális plató                        HA
        L2 = [X,X|_],           % L2 két azonos X elemmel kezdődő lista  ÉS
        append([L1,L2,L3], L),  % L szétszedhető L1, L2, L3 részlistákra ÉS
        forall( member(Y, L2),  % L2 minden Y eleme
                X = Y ),        %    azonos X-szel                       ÉS
        \+ L3 = [X|_],          % L3 nem X-szel kezdődő lista  (1)       ÉS
        \+ last(L1, X),         % L1 nem X-szel végződő lista  (2)       ÉS
        length(L2, Len),        % L2 hossza Len,                         ÉS
        length([a|L1], I).      % I = 1+(L1 hossza).

% Vegyük észre, hogy az (1) ill. (2) feltételek akkor is teljesülnek, ha
% az L3 ill. L1 listák üresek!

% Az alábbi eljárások segítségével a fenti plato0/4 eljárással ekvivalens,
% de lényegesen hatékonyabb plato/4 predikátumot valósítunk majd meg.


% 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 elejerol leszedheto k db A es
% marad egy nem A-val kezdodo M, tovabba 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 segédeljárás nélküli.
pl_kezdetu1([A,A], 2, []).
pl_kezdetu1([A|L], Len, M) :-
        L = [A|L1],
        L1 = [B|_],
        (  B = A -> pl_kezdetu1(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)
    ).

%   % elso_plato(L, Len, X, M): Az L listában van plató, Az első maximális
%   % plató Len hosszúságú és X elemekből képzett, továbbá ezt a platót egy
%   % M maradéklista követi.
%
%   | ?- elso_plato([a,b,b,b,b,a,a,c,b,b], Len, X, M).
%   Len = 4, X = b, M = [a,a,c,b,b] ? ; no

elso_plato(L, Len, X, M) :-
        (   pl_kezdetu(L, Len, M) ->
            L = [X|_]
        ;   L = [_|L1],
            elso_plato(L1, Len, X, M)
        ).

% Nem hatékony, mert kétszer hívja elso_plato/4-et
plato1(L, Len, X) :-
        elso_plato(L, Len, X, _M).
plato1(L, Len, X) :-
        elso_plato(L, _Len, _X, M),
        plato1(M, Len, X).

% plato1 ekvivalens átírása olyan formába, hogy a két klóz pontosan
% ugyanazzal az elso_plato hívással kezdődjék:
plato1a(L, Len, X) :-
        elso_plato(L, Len0, X0, _M0),
        Len = Len0, X = X0.
plato1a(L, Len, X) :-
        elso_plato(L, _Len0, _X0, M0),
        plato1a(M0, Len, X).
```

```
% plato1a-val logikailag ekvivalens, de lényegesen hatékonyabb változat:
plato2(L, Len, X) :-
        elso_plato(L, Len0, X0, M0),
        (   Len = Len0, X = X0
        ;   plato1a(M0, Len, X)
        ).



plato3(L, Len, X) :-        % Az L listában található egy Len hosszúságú,
                            % X elemekből képzett maximális plató        HA
    L2 = [X,X|_],           % L2 két azonos X elemmel kezdődő lista      ÉS
    append(L1, L23, L),     % L szétszedhető L1 és L23 részlistákra      ÉS
    append(L2, L3, L23),    % L23 szétszedhető L2 és L3 részlistákra     ÉS
    minden_eleme(L2, X),    % Az L2 lista minden eleme X                 ÉS
    \+ L3 = [X|_],          % L3 nem X-szel kezdődő lista  (1)           ÉS
    \+ last(L1, X),         % L1 nem X-szel végződő lista  (2)           ÉS
    length(L2, Len).        % L2 hossza Len.

% Vegyük észre, hogy az (1) ill. (2) feltételek akkor is teljesülnek, ha
% az L3 ill. L1 listák üresek!

% minden_eleme(L, X): Az L lista minden eleme X-szel egyenlő.
minden_eleme([], _).
minden_eleme([X|L], X) :-
        minden_eleme(L, X).


%
% 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)
    ).
```