```
%               Deklaratív Programozás gyakorlat 2012.11.09
%                    Prolog programozás: listák, univ
%                              MEGOLDÁSOK
% ---------------------------------------------------------------


% Listakezeléssel kapcsolatos feladatok
% =====================================

% 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([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

%     Használjon feltételes szerkezetet!

% 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([], 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).

% ------------------------------------------------------------------------------

%     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
.)

%     Például az  [a-b,a-c], [a-c,b-a], [b-a,a-c], [c-a,a-b] stb.  mind
%     ugyanazt a (matematikai értelemben vett) gráfot írják le.

%     Az [a-b,a-c,b-c,b-d,b-e,c-d,c-e,d-e] kifejezés az alábbi gráfot írja le
:

%                    a
%                   / \
%                  /   \
%                 /     \
%               b-------c
%               |\     /|
%               |  \ /  |
%               |  / \  |
%               |/     \|
%               d-------e

%     Gyerekkorukban találkozhattak azzal a feladattal, hogy ezt a gráfot egy
%     folytonos vonallal rajzolják meg.

%     Egy 'graph' típusú [a1-b1,a2-b2,...,an-bn] Prolog listát folytonos
%     vonalnak hívunk, ha b1=a2, b2=a3, ...,    b(n-1) = an.
%

% 2.  Írja meg az alábbi fejkommentnek  megfelelő  draw/2 Prolog eljárást

%     % 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.

%     Törekedjék minél egyszerűbb megoldásra, nem kell a hatékonysággal
%     foglalkoznia. Használhatja a lists könyvtár eljárásait.

%     | ?- 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

:- 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([], []).
draw(G, [E|L]) :-
        select_edge(E, G, G1),
        (   G1 == [] -> L = []
        ;   adjacent(E, L),
            draw(G1, L)
        ).

% select_edge(E, G, G1): A G irányítatlan gráfból elhagyható az E él és
% marad a G1 gráf.
select_edge(E, G, G1) :-
```

```
        select(E0, G, G1),
        same_edge(E0, E).

% same_edge(E, E1): E és E1 ugyanazt az irányítatlan élet ábrázolja.
same_edge(E, E).
same_edge(A-B, B-A).

% adjacent(E, G): Az E él végpontja azonos a G gráf első élének
% kezdőpontjával.
adjacent(_-B, [B-_|_]).

% draw1(+G, -L): ugyanaz, mint draw/2, csak segédeljárás nélkül.
draw1([], []).
draw1(G, [A-B|L]) :-
        select(E, G, G1),
        (   E = A-B
        ;   E = B-A
        ),
        (   G1 == [] -> L = []
        ;   L = [B-_|_],
            draw1(G1, L)
        ).


% 3.  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.  (szorgalmi 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).


% Meta-logikai beépített eljárással kapcsolatos feladatok
% ======================================================


% 5.  Atomok szeletelése
%
%     Egy A atom szuffixumának nevezünk egy S atomot, ha S az A utolsó
%     valahány karakterét tartalmazza, az A-beli sorrend megtartásával.
%
%     % atom_suffix(+Atom, ?Suffix, +Before): Az Atom névkonstansból a Suffix

%     % atom úgy áll elő, hogy A elejéről elhagyunk Before számú karaktert.
%     | ?- atom_suffix(abcde, Suffix, 3).
%     Suffix = de ? ;
%     no
%
%     (Nem használhatja a sub_atom/5 beépített eljárást.)

atom_suffix(Atom, Suffix, Before) :-
        atom_codes(Atom, Codes),
        drop(Before, Codes, Suffix_Codes),
        atom_codes(Suffix, Suffix_Codes).

% drop(+N, +L0, -L): Az L listát úgy kapjuk, hogy az L0 lista első N elemét
% elhagyjuk, ahol N nem-negatív egész.
drop(0, L, L).
drop(N, [_X|L0], L) :-
        N > 0, N1 is N-1,
        drop(N1, L0, L).



% 6.  Általános Prolog kifejezés részkifejezéseinek vizsgálata
```

```
%     % mern(+K, +N): A K általános Prolog kifejezésben előforduló összes egész
%     % szám határozottan nagyobb mint N (mern = minden egész részkifejezése
%     % nagyobb mint)

%     | ?- mern(1, 1).
%     no
%     | ?- mern(1, 0).
%     yes
%     | ?- mern(f(X,[1,3,b],g(2,1,a0)), 0).
%     true ? ;
%     no
%     | ?- mern(f(X,[1,3,b],g(2,1,a0)), 1).
%     no

%     Megjegyzések:

%     a. A "K1 Prolog kifejezésben előfordul a K2 kifejezés" relációt
%        reflexívnek tekintjük, azaz egy K kifejezésben önmaga mindenképpen
%        előfordul. Ez a megjegyzés vonatkozik az ezután következő
%        feladatokra is.

%     b. Vigyázzon arra, hogy a kifejezésben változók is előfordulhatnak.

% mern(+K, +N): A K általános Prolog kifejezésben előforduló összes egész
% szám határozottan nagyobb mint N (mern = minden egész részkifejezése
% nagyobb mint)
mern(K, N) :-
        (   integer(K) ->
            K > N
        ;   compound(K) ->
            K =.. [_Fun|Args],
            mern_lista(Args, N)
        ;   true
        ).

% mern_lista(+Ks,+N): a Ks listában szereplo minden kifejezésben minden
% egész nagyobb, mint N.
mern_lista([], _N).
mern_lista([K|Ks], N) :-
        mern(K,N),
        mern_lista(Ks, N).

% 7.  Általános Prolog kifejezés bizonyos részkifejezéseinek felsorolása

%     % reszatom(+K, ?A): A a K általános Prolog kifejezésben előforduló atom.

%     | ?- reszatom(a, X).
%     X = a ? ;
%     no
%     | ?- reszatom(f(X,[1,3,b],g(2,1,a0)), A).
%     A = b ? ;
%     A = [] ? ;
%     A = a0 ? ;
%     no

%     Megjegyzés: a struktúranevet nem tekintjük a struktúrakifejezés
%     részének.

% reszatom(+K, ?A): A a K általános Prolog kifejezésben előforduló atom.
reszatom(K, A) :-
        (   atom(K) ->
            K = A
        ;   compound(K) ->
            K =.. [_Fun|Args],
            member(Arg, Args),
            reszatom(Arg, A)
        ).
```

```
% 8.  Általános Prolog kifejezés bizonyos részkifejezéseinek akkumulálása

%     % osszege(+K, ?Ossz): Ossz a K kifejezésben előforduló egész számok
%     % összege.

%     | ?- osszege(a, S).
%     S = 0 ? ;
%     no
%     | ?- osszege(1, S).
%     S = 1 ? ;
%     no
%     | ?- osszege(f(X,[1,3,b],g(2,1,a0)), S).
%     S = 7 ? ;
%     no

% osszege(+K, ?Ossz): Ossz a K kifejezésben előforduló egész számok
% összege.
osszege(K, Sum) :-
        (   integer(K) ->
            Sum = K
        ;   compound(K) ->
            K =.. [_Fun|Args],
            osszege_lista(Args, 0, Sum)
        ;   Sum = 0
        ).

% osszege_lista(+KL, +Ossz0, ?Ossz): A KL listában előforduló egész számok
% összege plusz az Ossz0 szám egyenlő az Ossz számmal.
osszege_lista([], Sum, Sum).
osszege_lista([X0|L0], Sum0, Sum) :-
        osszege(X0, Sum1),
        Sum2 is Sum0+Sum1,
        osszege_lista(L0, Sum2, Sum).

% 9.  Általános Prolog kifejezés bizonyos részkifejezéseinek átalakítása

% roviditett(+Kif0, ?Kif): Kif a Kif0 általános Prolog kifejezésből úgy áll
% elő, hogy minden, benne előforduló egész számot egy eggyel nagyobb
% számra cserélünk.
roviditett(Kif0, Kif) :-
        (   atom(Kif0), Kif0 \== '' ->
            atom_suffix(Kif0, Kif, 1)
        ;   compound(Kif0) ->
            Kif0 =.. [Fun|Args0],
            roviditett_lista(Args0, Args),
            Kif =.. [Fun|Args]
        ;   Kif = Kif0
        ).

% roviditett_lista(+L0, ?L): L az L0 listából úgy áll elő, hogy minden benne
% előforduló egész számot egy eggyel nagyobb számra cserélünk.
roviditett_lista([], []).
roviditett_lista([X0|L0], [X|L]) :-
        roviditett(X0, X),
        roviditett_lista(L0, L).
```