```
-module(dp10a_gy6).
-author('patai@iit.bme.hu, hanak@inf.bme.hu').
-vsn('2010-11-27').
%-export([]).
-compile(export_all).

%%                   Deklaratív programozás, 6. gyakorlat
%%                   Erlang programozás: listák használata
%%                            2010. 11. 22. & 24.
%%
%% --------------------------------------------------------------------------
%%
%% Írjon olyan Erlang-eljárást, amely megfelel az adott fejkommentnek.  Az
%% eljárásban felhasználhatja a már megoldott feladatokhoz megírt eljárásait.
%%
%%
%% --------------------------------------------------------------------------
%% 1. Listában párosával elõforduló elemek listája

%% @spec parban(Xs::[term()]) -> Zs::[term()].
%% A Zs lista az Xs lista összes olyan elemét tartalmazza, amelyet
%% vele azonos értékû elem követ.

parban_1([E,E|Xs]) ->
    [E|parban_1([E|Xs])];
parban_1(Xs) ->
    if
        length(Xs) > 1 ->
            parban_1(tl(Xs));
        true ->
            []
    end.

%% Alternatív megoldás: parban/1 listanézettel.
parban_2(Xs) ->
    [E || [E,E|_] <- tails_1(Xs)].

%% parban([a,a,a,2,3,3,a,2,b,b,4,4]) == [a,a,3,b,4].


%% --------------------------------------------------------------------------
%% 2. Lista egyre rövidülõ szuffixumainak listája

%% @spec tails(Ls::[term()]) -> Zss::[[term()]].
%% A Zss lista az Ls listát és egyre rövidülõ szuffixumait tartalmazza.

%% dp10s-gy3.txt-ben van hasonló feladat listaSzuffixumai néven.

tails_1([]) ->
    [[]];
tails_1(L=[_|Ls]) ->
    [L|tails_1(Ls)].

%% Alternatív megoldás: tails/1 foldr-rel.
tails_2(Ls) ->
    lists:foldr(fun(X,Xs) -> [[X|hd(Xs)]|Xs] end, [[]], Ls).

%% tails([1,4,2]) == [[1,4,2],[4,2],[2],[]].
%% tails([a,b,c,d,e]) == [[a,b,c,d,e],[b,c,d,e],[c,d,e],[d,e],[e],[]].
```



```
%% --------------------------------------------------------------------------
%% 3. Listában párosával elõforduló részlisták listája

%% @spec dadogo(Ls::[term()]) -> Dss::[[term()]].
%% A Dss lista az Ls lista összes olyan nemüres (folytonos) részlistáját
%% tartalmazza, amelyet vele azonos értékû részlista követ.

dadogo(Ls) ->
    [Ds || Rs = [_,_|_] <- tails_1(Ls),
           N <- lists:seq(1,length(Rs) div 2),
           begin
               {Ds,Zs} = lists:split(N,Rs),
               lists:prefix(Ds,Zs)
           end].

%% dadogo([a,a,a,2,3,3,a,2,b,b,b,b]) == [[a],[a],[3],[b],[b,b],[b],[b]].
%% dadogo([1,2,1,2,c,a,a,c]) == [[1,2],[a]].


%% --------------------------------------------------------------------------
%% 4. Lista elsõ monoton növekvõ részlistája (futama)

%% @spec rampa(Ls::[term()]) -> {Zs::[term()], Ms::[term()]}.
%% A Zs lista az Ls lista elsõ monoton növekvõ futama, az Ms az Xs maradéka.

rampa_1(Ls) ->
    rampa_1(Ls, []).

%%%% Segédfüggvény gyûjtõargumentummal.
%% @spec rampa(Xs::[term()], Rs::[term()]) -> {Zs::[term()],Ms::[term()]}.
%% A Zs lista az Xs lista elsõ monoton növekvõ futama az Rs fordítottja
%% mögé fûzve; az Ms az Xs maradéka (Zs utáni része).

rampa_1([], Rs) ->
    {lists:reverse(Rs), []};
rampa_1([X], Rs) ->
    {lists:reverse([X|Rs]), []};
rampa_1([X1,X2|Xs], Rs) when X1 > X2 ->
    {lists:reverse([X1|Rs]),[X2|Xs]};
rampa_1([X|Xs], Rs) ->
    rampa_1(Xs, [X|Rs]).

rampa_2([]) ->
    {[],[]};
rampa_2(Ls) ->
    Rs = lists:takewhile(fun({X,Y}) -> X =< Y end,
                         lists:zip(lists:sublist(Ls,length(Ls)-1),tl(Ls))),
    lists:split(length(Rs)+1,Ls).

%% Alternatív megoldás tails/1 és lists:splitwith/2 használatával.
%%
%% rampa_3/1-ben tails/1 eredménye az Ls egyre rövidülõ farkainak a listája, pl.
%% tails([a,b,c,b,a]) == [a,b,c,b,a]
%%                         [b,c,b,a]
%%                           [c,b,a]
%%                             [b,a]
%%                               [a]
%%                                []
%% splitwith/2 közülük azokat adja vissza Rss-ben, amelyeknek a feje nem nagyobb
%% a második elemüknél, Mss-ben pedig - az elsõ kivételével - azokat amelyekre
%% ez nem áll fenn, pl.
%%    Rss == [[a,b,c,b,a],[b,c,b,a]]
%%    Mss == [[b,a],[a],[]]
%%
```

```
%% Ha ezek után az Rss lista üres, akkor egyedül Ls feje "képez" monoton növekvõ
%% sorozatot, az Ls farka pedig a maradék.
%%
%% Ha az Rss nem üres, akkor elsõ részlistájának a fejébõl és összes
%% részlistájának a második elemébõl képezzük a monoton növekvõ sorozatot (azaz
%% a futamot, az eredménypár elsõ tagját), az Mss minden nemüres részlistájának
%% az elsõ eleméből pedig a maradéklistát (az eredménypár második tagját).
%%
rampa_3([]) ->
    {[],[]};
rampa_3(Ls) ->
    F = fun([X1,X2|_]) -> X1 =< X2; (_) -> false end,
    {Rss,[_|Mss]} = lists:splitwith(F, tails_1(Ls)),
    case Rss of
        [] ->
            lists:split(1,Ls);
        [Es|_] ->
            % {futam (eredménypár 1. tagja), maradéklista (eredménypár 2. tagja}
            {[hd(Es)|[X || [_,X|_] <- Rss]], [X || [X|_] <- Mss]}
    end.

%%%% splitwith(Pred, List) -> {takewhile(Pred, List), dropwhile(Pred, List)}.
%% lists:splitwith(fun erlang:is_atom/1, [a,b,c,d,1,2,3,4,e,f,5,6,7]) ==
%%    {[a,b,c,d],[1,2,3,4,e,f,5,6,7]}.

%% rampa([1,2,2,3,2,4,5,6,6,6,7,6,8,2,3,3,4,5,6,0,6,5,4,3,2,1]) ==
%%   {[1,2,2,3],[2,4,5,6,6,6,7,6,8,2,3,3,4,5,6,0,6,5,4,3,2,1]}.
%% rampa([a,b,b,c,b,b,a]) == {[a,b,b,c],[b,b,a]}.


%% ----------------------------------------------------------------------------
%% 5. Lista monoton növekvõ részlistáinak (futamainak) listája

% @spec rampak(Xs::[term()]) -> Xss::[[term()]].
% Az Xss az Xs monoton növekvõ futamainak listája.

rampak_1([]) ->
    [];
rampak_1(Xs) ->
    {Rs,Ms} = rampa_1(Xs),
    [Rs|rampak_1(Ms)].

%% Alternatív megoldás: rampak/1 slash függvénnyel.
rampak_2(Xs) ->
    slash(fun rampa_1/1,Xs).

%% rampak([1,2,2,3,2,4,5,6,6,6,7,6,8,2,3,3,4,5,6,0,6,5,4,3,2,1]) ==
%%   [[1,2,2,3],[2,4,5,6,6,6,7],[6,8],[2,3,3,4,5,6],[0,6],[5],[4],[3],[2],[1]].


%% ----------------------------------------------------------------------------
%% 6. Lista darabokra szabdalása

%% @spec slash(F::fun([term()]) -> {term(),[term()]}, Xs::([term()]))
%%                                                  -> Zs::([term()]).
%% A Zs lista az Xs listából az F ismételt alkalmazásával elõálló lista,
%% ahol F párban visszaadja a Zs következõ elemét és az Xs még nem
%% feldolgozott részét.

slash(_F,[]) ->
    [];
slash(F,Xs) ->
    {Rs,Ms} = F(Xs),
    [Rs|slash(F,Ms)].
```



```
%% slash(fun(Xs) -> lists:split(3, Xs) end, "jaaaj!!! nem jooo!") ==
%%   ["jaa","aj!","!! ","nem"," jo","oo!"].


%% ----------------------------------------------------------------------------
%% 7. Lista számtani sorozatot alkotó prefixuma

%% @spec dif(Ls::[number()]) -> {Ds::[number()], Zs::[number()]}.
%% A Ds lista az Ls lista számtani sorozatot alkotó prefixuma, a Zs az Ls
%% maradéka (Zs utáni része).

dif_1(Ls) ->
    if
        length(Ls) < 2 ->
            {Ls,[]};
        true ->
            [X1,X2|_] = Ls,
            dif(Ls, X2-X1, [])
    end.

%% @spec dif(Ls::[number()],N::number(),Rs::[number()]) ->
%%                                        {Ds::[number()], Zs::[number()]}.
%% A Ds lista az Ls lista N különbségû számtani sorozatot alkotó prefixuma
%% az Rs lista fordítottja mögé fûzve, a Zs az Ls maradéka (Zs utáni része).

dif([], _, Rs) ->
    {lists:reverse(Rs), []};
dif([X], _, Rs) ->
    {lists:reverse([X|Rs]), []};
dif([X1,X2|Xs], N, Rs) when X2-X1 /= N ->
    {lists:reverse([X1|Rs]),[X2|Xs]};
dif([X|Xs], N, Rs) ->
    dif(Xs, N, [X|Rs]).

%% Alternatív megoldás: dif/1 feltétel helyett mintaillesztéssel.
dif_2([X1,X2|_]=Ls) ->
    dif(Ls, X2-X1, []);
dif_2(Ls) -> {Ls,[]}.

%% dif([1,2,3,4,8,16,32,33,34]) == {[1,2,3,4],[8,16,32,33,34]}.
%% dif([1,2,3,4,8,16,24,32,33,34]) == {[1,2,3,4],[8,16,24,32,33,34]}.


%% ----------------------------------------------------------------------------
%% 8. Lista számtani sorozatot alkotó részlistáinak listája

%% @spec difek(Xs::[number()]) -> Dss::[[number()]].
%% A Dss lista az Xs lista számtani sorozatot alkotó részlistáinak listája.

difek_1([]) ->
    [];
difek_1(Xs) ->
    {Rs,Ms} = dif_1(Xs),
    [Rs|difek_1(Ms)].

%% Alternatív megoldás: difek/1 slash függvénnyel.
difek_2(Xs) ->
    slash(fun dif_2/1,Xs).

%% difek([1,2,3,4,8,16,32,33,34]) == [[1,2,3,4],[8,16],[32,33,34]].
%% difek([1,2,3,4,8,16,24,32,33,34]) == [[1,2,3,4],[8,16,24,32],[33,34]].
```

```
%% Feladat: úgy módosítani dif/3-at, hogy ha egy szám az egyik
%% számtani sorozat utolsó és egyben a következõ elsõ eleme is
%% lehet, akkor mindkettõben vegyük figyelembe.
%% NB. Ilyenkor egy sorozat záróeleme mindenképpen egy következõ
%% sorozat kezdõeleme is lesz egyben, hiszen már két elem is
%% számtani sorozatot alkot.

dif1([], _, Rs) ->
    {lists:reverse(Rs), []};
dif1([X], _, Rs) ->
    {lists:reverse([X|Rs]), []};
dif1([X1,X2|Xs], N, Rs) when X2-X1 /= N ->
    {lists:reverse([X1|Rs]),[X1,X2|Xs]};
dif1([X|Xs], N, Rs) ->
    dif1(Xs, N, [X|Rs]).

dif1([X1,X2|_]=Ls) ->
    dif1(Ls, X2-X1, []);
dif1(Ls) -> {Ls,[]}.

%% slash(fun dif1/1,[1,2,3,4,5,6,7,8,16,24,32,33,34]) ==
%%   [[1,2,3,4,5,6,7,8],[8,16,24,32],[32,33,34]].
%% slash(fun dif1/1,[1,2,3,4,5,6,7,16,24,32,33,34]) ==
%%   [[1,2,3,4,5,6,7],[7,16],[16,24,32],[32,33,34]].


%% ----------------------------------------------------------------------------
%% 9. Keresõfa adott tulajdonságú csomópontjainak listája

%% @type btree() = e | {n,int(),btree(),btree()}.

%% @spec nagyobbak(T::btree(), E::int()) -> Rs::[btree()].
%% A T keresõfa-tulajdonságú fa E-nél nagyobb értéket tartalmazó
%% csomópontjainak listája Rs.

nagyobbak(e,_) ->
    [];
nagyobbak(T={n,X,L,R},E) when X > E ->
    nagyobbak(L,E) ++ [T] ++ s(R);
nagyobbak({n,_X,_L,R},E) ->
    nagyobbak(R,E).

%% @spec s(T::btree()) -> Rs::[btree()].
%% A T keresõfa-tulajdonságú fa csomópontjainak listája Rs.

s(e) ->
    [];
s(N={n,_X,L,R}) ->
    s(L) ++ [N] ++ s(R).

%%                    3
%%              2           6
%%           1   e       4   9
%%          e e         e e e e

%% nagyobbak({n,3,{n,2,{n,1,e,e},e},{n,6,{n,4,e,e},{n,9,e,e}}},2) ==
%%   [{n,3,{n,2,{n,1,e,e},e},{n,6,{n,4,e,e},{n,9,e,e}}},
%%    {n,4,e,e},
%%    {n,6,{n,4,e,e},{n,9,e,e}},
%%    {n,9,e,e}].
```



```
%% @spec nagyobbak_utja(T::btree(), E::int()) -> Rs::[btree()]
%% Az Rs {V, Bs} párokból álló lista, ahol V a T bináris fa valamely
%% csomópontjában található, E-nél nagyobb érték, Bs pedig a V érték
%% elérési útvonalát a b(alra), illetve j(obbra) atomokkal jelzõ lista.

nagyobbak_utja_1(e,_) ->
    [];
nagyobbak_utja_1({n,X,B,J}, E) ->
    Uk = fun() ->
                [{Y,[b|U]} || {Y,U} <- nagyobbak_utja_1(B, E)] ++
                [{Y,[j|U]} || {Y,U} <- nagyobbak_utja_1(J, E)]
         end,
    if
        X > E ->
            [{X,[]}|Uk()];
        true ->
            Uk()
    end.

%% Itt nem igazán kell trükközni a késõi kiértékeléssel, mint a filter
%% esetén, úgyhogy egyszerûsíthetõ a kód:

nagyobbak_utja_2(e,_) ->
    [];
nagyobbak_utja_2({n,X,B,J}, E) ->
    Uk = [{Y,[b|U]} || {Y,U} <- nagyobbak_utja_2(B, E)] ++
         [{Y,[j|U]} || {Y,U} <- nagyobbak_utja_2(J, E)],
    if
        X > E ->
            [{X,[]}|Uk];
        true ->
            Uk
    end.

%%                    0
%%                1           6
%%            3   e       4   9
%%           e e         e e e e

%% nagyobbak_utja({n,0,{n,1,{n,3,e,e},e},
%%                     {n,6,{n,4,e,e},{n,9,e,e}}},2) ==
%%   [{3,[b,b]},{6,[j]},{4,[j,b]},{9,[j,j]}].

%%                    0
%%                1           6
%%            3   e       4       9
%%           e e         e e   7   8
%%                           e e e e

%% nagyobbak_utja({n,0,{n,1,{n,3,e,e},e},
%%                        {n,6,{n,4,e,e},{n,9,{n,7,e,e},{n,8,e,e}}}},2) ==
%%   [{3,[b,b]},{6,[j]},{4,[j,b]},{9,[j,j]},{7,[j,j,b]},{8,[j,j,j]}].
```