```erlang
-module('dp19a-gy5').
-compile(export_all).
-author('patai@iit.bme.hu, hanak@emt.bme.hu, kapolnai@iit.bme.hu').
-vsn('$LastChangedDate: 2019-11-05 09:17:30 +0100 (k, 05 nov 2019) $$').

%%------------------------------------------------------------------------
%%                            BINÁRIS FÁK
%%------------------------------------------------------------------------

-type fa()      :: level | {any(),fa(),fa()}.
-type egeszfa() :: level | {integer(),egeszfa(),egeszfa()}.


% 1.
-spec fa_noveltje(F0::egeszfa()) -> F::egeszfa().
% Az F fa minden címkéje eggyel nagyobb az F0 azonos helyen lévő címkéjénél.
fa_noveltje(level) ->
    level;
fa_noveltje({C,Bfa,Jfa}) ->
    {C+1, fa_noveltje(Bfa), fa_noveltje(Jfa)}.

% 2.
-spec fa_tukorkepe(F0::fa()) -> F::fa().
% F az F0 fa tükörképe.
fa_tukorkepe(level) ->
    level;
fa_tukorkepe({C,Bfa,Jfa}) ->
    {C, fa_tukorkepe(Jfa), fa_tukorkepe(Bfa)}.


% 3.
-spec inorder(F::fa()) -> L::list().
% L az F fa elemeinek a fa inorder bejárásával létrejövő listája.
inorder(level) ->
    [];
inorder({C,Bfa,Jfa}) ->
     inorder(Bfa) ++ [C|inorder(Jfa)].

-spec preorder(F::fa()) -> L::list().
% L az F fa elemeinek a fa preorder bejárásával létrejövő listája.
preorder(level) ->
    [];
preorder({C,Bfa,Jfa}) ->
     [C|preorder(Bfa) ++ preorder(Jfa)].

-spec postorder(F::fa()) -> L::list().
% L az F fa elemeinek a fa postorder bejárásával létrejövő listája.
postorder(level) ->
    [];
postorder({C,Bfa,Jfa}) ->
    postorder(Bfa) ++ (postorder(Jfa) ++ [C]).


% 4.
-spec tartalmaz(C::any(), F::fa()) -> B::boolean().
% B igaz, ha C az F fa valamely címkéje.
tartalmaz(_, level) ->
    false;
tartalmaz(C, {C,_,_}) ->
    true;
tartalmaz(C, {_,Bfa,Jfa}) ->
    tartalmaz(C, Bfa) orelse tartalmaz(C, Jfa).


% 5.
-spec elofordul(C::any(), F::fa()) -> N::integer().
% A C címke az F fában N-szer fordul elő.
elofordul(_, level) ->
    0;
elofordul(C, {R,Bfa,Jfa}) ->
    if C =:= R -> 1;
       true    -> 0
    end
        + elofordul(C, Bfa) + elofordul(C, Jfa).


% 6.
-spec cimkek(F::fa()) -> L::[any()].
% L az F címkéinek listája inorder sorrendben.
cimkek(Fa) -> cimkek(Fa, []).

-spec cimkek(F::fa(), L0::[any()]) -> L::[any()].
% L az F címkéinek listája inorder sorrendben L0 elé fűzve.
cimkek(level, L) ->
    L;
cimkek({R,Bfa,Jfa}, L) ->
    cimkek(Bfa, [R|cimkek(Jfa, L)]).


% 7.
-spec fa_balerteke(F::fa()) -> {ok, C::any()} | error.
% Egy nemüres F fa bal oldali szélső címkéje C (minden
% felmenőjére is  igaz, hogy bal oldali gyermek).
fa_balerteke(level) ->
    error;
fa_balerteke({C,level,_}) ->
    {ok, C};
fa_balerteke({_,Bfa,_}) ->
    fa_balerteke(Bfa).


% 8.
-spec fa_jobberteke(F::fa()) -> {ok, C::any()} | error.
% Egy nemüres F fa jobb oldali szélső címkéje C (minden
% felmenőjére is igaz, hogy jobb oldali gyermek).
fa_jobberteke(level) ->
    error;
fa_jobberteke({C,_,level}) ->
    {ok, C};
fa_jobberteke({_,_,Jfa}) ->
    fa_jobberteke(Jfa).


% 9. a)
-spec rendezett_fa(F::fa()) -> B::boolean().
% B igaz, ha az F fa rendezett.
rendezett_fa(F) ->
    L = cimkek(F),
    L =:= lists:sort(L).
```

```erlang
% 9. b)
-spec rendezett_fa_2(F::fa()) -> B::boolean().
% B igaz, ha az F fa rendezett.
rendezett_fa_2(level) ->
    true;
rendezett_fa_2({C,Bfa,Jfa}) ->
    case fa_jobberteke(Bfa) of
        error  ->
            true;
        {ok,J} ->
            J < C
    end
      andalso
    rendezett_fa_2(Bfa)
      andalso
    case fa_balerteke(Jfa) of
        error ->
            true;
        {ok,B} ->
            C < B
    end
      andalso
    rendezett_fa_2(Jfa).


% 10.
-type ut() :: [any()].
-spec utak(F::fa()) -> CimkezettUtak::[{C::any(), CU::ut()}].
% A CimkezettUtak lista az F fa minden csomópontjához egy {C,CU} párt
% társít, ahol C az adott csomópont címkéje, CU pedig az adott
% csomóponthoz vezető útvonal.
utak(Fa) -> utak(Fa, []).

-spec utak(F::fa(), Eddigi::ut()) -> CimkezettUtak::[{C::any(), U::ut()}].
% A CimkezettUtak lista az F fa minden csomópontjához egy {C,U} párt
% társít, ahol C az adott csomópont címkéje, U pedig az adott
% csomóponthoz vezető útvonal az Eddigi eddigi útvonal elé fűzve.
utak(level, _) ->
    [];
utak({C,Bfa,Jfa}, Eddigi) ->
    %  Eddigi1 = Eddigi ++ [C], % Költséges művelet: O(n2)!
    Eddigi1 =  lists:reverse([C | lists:reverse(Eddigi)]), % Kevésbé: O(2n).
    [{C, Eddigi} | utak(Bfa, Eddigi1)] ++ utak(Jfa, Eddigi1).


% 11. a)
-spec cutak(C::any(), F::fa()) -> Utak::[{C::any(), CU::ut()}].
% Utak azon csomópontok útvonalainak listája F-ben, amelyek címkéje C.
cutak(C, Fa) ->
    [CU || {C0,_} = CU <- utak(Fa), C0 =:= C].

% 11. b)
-spec cutak_2(C::any(), F::fa()) -> Utak::[{C::any(), CU::ut()}].
% Utak azon csomópontok útvonalainak listája F-ben, amelyek címkéje C.
cutak_2(C, Fa) -> cutak_2(C, Fa, []).

-spec cutak_2(C::any(), F::fa(), Eddigi::ut()) ->
                     CimkezettUtak::[{C::any(), U::ut()}].
% A CimkezettUtak lista az F fa minden C címkéjű csomópontjához egy {C,U}
% párt társít, ahol C az adott csomópont címkéje, U pedig az adott
% csomóponthoz vezető útvonal az Eddigi eddigi útvonal elé fűzve.
cutak_2(_, level, _) ->
    [];
cutak_2(C, {R,Bfa,Jfa}, Eddigi) ->
    Eddigi1 = Eddigi ++ [R],
    Cutak = cutak_2(C, Bfa, Eddigi1) ++ cutak_2(C, Jfa, Eddigi1),
    if R =:= C -> [{C, Eddigi} | Cutak];
       true    -> Cutak
    end.

```

```erlang
%-----------------------------------------------------------------------------
%                                LUSTA FARKÚ LISTA
%-----------------------------------------------------------------------------

-type lazy_list() :: nil | {any(), fun(() -> lazy_list())}.
% Ez a lista félig lusta: a fej mindig kiértékelődik, a farok lusta.

-spec infseq(N::integer()) -> LL::lazy_list().
% Az N-nel kezdődő, egyesével növekedő egész számok lusta farkú listája LL.
infseq(N) -> {N, fun() -> infseq(N+1) end}.

% 12. a)
-spec nth(L::lazy_list(), N::integer()) -> X::any().
% Az L lusta lista N-edik eleme X (számozás 1-től).
nth({H,_T}, 1) ->
    H;
nth({_H,T}, N) ->
    nth(T(), N-1).

% 12. b)

-spec nth_2(L::lazy_list(), N::integer()) -> {ok, X::any()} | error.
% Az L lusta lista N-edik eleme X (számozás 1-től). A visszatérési
% érték az 'error' atom, ha az L lista üres, vagy ha N < 1.
nth_2({H,_T}, 1) ->
    {ok, H};
nth_2({_H,T}, N) when N > 1 ->
    nth_2(T(), N-1);
nth_2(_, _) ->
    error.


% 13. a)
-spec fibs(Curr::integer(), Next::integer()) -> LL::lazy_list().
% A 0. és 1. Fibonacci-számokkal (Curr =:= 0, Next =:= 1) meghívva
% fibs-et, LL a Fibonacci-számokat tartalmazó lusta lista.
fibs(Curr, Next) ->
    {Curr, fun() -> fibs(Next, Curr+Next) end}.

% 13. b)
-spec fib(N::integer()) -> F::integer().
% F az N-edik Fibonacci-szám.

fib(N) ->
    nth(fibs(0,1), N+1).


```

```erlang
%-------------------------------------------------------------------------
%                                NZH Erlang (mintafeladatok)
%-------------------------------------------------------------------------


% 14.
is_space(X) -> X =:= 32.

mitirki() ->
   X1 = {{some,6}, [$A], length([fun erlang:'element'/2]),
         (fun erlang:'>'/2)(2,3)},
   X2 = lists:map(fun is_space/1, lists:concat(["xx7","a5","8z"])),
   F = fun lists:seq/2,
   X3 = [F(X, Y) || X <- F(1, 2), Y <- lists:reverse(F(1, 3)), X =< Y],
   [ X1 =:= {{some,6}, "A", 1, false},
     X2 =:= [false, false, false, false, false, false, false],
     X3 =:= [[1,2,3],[1,2],[1],[2,3],[2]]
   ].


% 15. a)
-spec szigetek(Xs::[integer()]) -> Rss::[[integer()]].
% Az Xs pozitív elemekből álló, folytonos, maximális
% hosszúságú részlistáinak listája az Rss lista.

szigetek(Xs) ->
    szigetek(lists:reverse(Xs), [], []).

szigetek([], [], Sss) ->
    Sss;
szigetek([], Ss, Sss) ->
    [Ss|Sss];
szigetek([X|Xs], Ss, Sss) when X > 0 ->
    szigetek(Xs, [X|Ss], Sss);
szigetek([_X|Xs], [], Sss) ->
    szigetek(Xs, [], Sss);
szigetek([_X|Xs], Ss, Sss) ->
    szigetek(Xs, [], [Ss|Sss]).

-spec sziget_2(Xs::[integer()], Zs::[integer()]) ->
                       {Ss::[integer()], Ms::[integer()]}.
% Ss az Xs pozitív elemekből álló, folytonos, maximális
% hosszúságú első részlistája Zs elé fűzve, Ms pedig az Xs maradéka.
sziget_2([], Zs) ->
    {Zs, []};
sziget_2([X|Xs], Zs) when X =< 0->
    {Zs, Xs};
sziget_2([X|Xs], Zs) ->
    sziget_2(Xs, [X|Zs]).

-spec szigetek_2(Xs::[integer()]) -> Rss::[[integer()]].
% Az Xs pozitív elemekből álló, folytonos, maximális
% hosszúságú részlistáinak listája az Rss lista.

szigetek_2([]) ->
    [];
szigetek_2([X|Xs]) when X =< 0 ->
    szigetek(Xs);
szigetek_2(Xs) ->
    {Ss, Ms} = sziget_2(Xs, []),
    [lists:reverse(Ss)|szigetek_2(Ms)].


```

```erlang
% 15. b)
-spec szfold(Zs::[integer()]) -> Ys::[{Hossz::integer(),Csucs::integer()}].
% A negatív számokat nem tartalmazó Zs egészlistában előforduló szárazföldek
% hosszát és legmagasabb pontját leíró {Hossz, Csucs} párok listája Ys.

szfold(Zs) ->
    F = fun (Xs) -> {length(Xs), lists:max(Xs)} end,
    lists:map(F, szigetek(Zs)).

-spec szfold_2(Zs::[integer()]) -> Ys::[{Hossz::integer(),Csucs::integer()}].
% A negatív számokat nem tartalmazó Zs egészlistában előforduló szárazföldek
% hosszát és legmagasabb pontját leíró {Hossz, Csucs} párok listája Ys.

szfold_2(Zs) ->
    F = fun (Xs) -> {length(Xs), lists:max(Xs)} end,
    lists:map(F, szigetek_2(Zs)).


% 16. a)
-spec dec2hex(D::integer()) -> H::string().
% A D decimális szám $0-$9 és $A-$F karakterek füzéreként ábrázolt
% hexadecimális megfelelője H.

dec2hex(0) ->
    [charcode(0)];
dec2hex(D) ->
    dec2hex(D, []).

dec2hex(0, Hs) ->
    Hs;
dec2hex(D, Hs) ->
    dec2hex(D div 16, [charcode(D rem 16)|Hs]).

-spec charcode(C::integer()) -> H::char().
% A 0 =< C =< 15 egésznek megfelelő hexadecimális jegy a
% $0, ..., $9, $A, ..., $F tartományban.
charcode(C) when C < 10 ->
    C+$0;
charcode(C) ->
    C-10+$A.


% 16. b)
-type fa(Elem) :: e | {n, Elem, fa(Elem), fa(Elem)}.
-spec d2hfa(Fa0::fa(integer())) -> Fa::fa(string()).
% A Fa0 decimális számokat tartalmazó fa hexadecimális számokat
% tartalmazó megfelelője Fa.

d2hfa(e) ->
    e;
d2hfa({n, D, Bfa, Jfa}) ->
    {n, dec2hex(D), d2hfa(Bfa), d2hfa(Jfa)}.

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


```

```
test(def) ->
    T0 = {1,
          {2,
           {4,level,level},
           {5,level,level}
          },
          {3,level,level}
         },
    T1 = {4,
          {3,level,level},
          {6,
           {5,level,level},
           {7,level,level}
          }
         },
    T2 = {a,
          {b, {v,level,level}, level},
          {c,
           level,
           {d,
            {w,{x,level,level},level},
            {f, {x,level,level},{y,level,level}
            }
           }
          }
         },
    T3 = {4,
          {3,
           {2,level,level},
           {1,level,level}
          },
          {6,
           {5,level,level},
           {7,level,level}
          }
         },
    {T0,T1,T2,T3}
;
test(fa1) ->
    {T0,T1,T2,_T3} = test(def),
    [
     fa_noveltje(T1) =:=
         {5,{4,level,level},{7,{6,level,level},{8,level,level}}},
     fa_tukorkepe(T1) =:=
         {4,{6,{7,level,level},{5,level,level}},{3,level,level}},
     inorder(T0) =:= [4,2,5,1,3],
     preorder(T0) =:= [1,2,4,5,3],
     postorder(T0) =:= [4,5,2,3,1],
     inorder(T1)   =:= [3,4,5,6,7],
     preorder(T1)  =:= [4,3,6,5,7],
     postorder(T1) =:= [3,5,7,6,4],
     tartalmaz(x, T1) =:= false,
     tartalmaz(x, T2) =:= true,
     elofordul(x, T1) =:= 0,
     elofordul(x, T2) =:= 2
    ]
;
```

```
test(fa2) ->
    {_T0,T1,T2,T3} = test(def),
    [
     cimkek(T1) =:= [3,4,5,6,7],
     fa_balerteke(T1) =:= {ok, 3},
     fa_balerteke(level) =:= error,
     fa_balerteke(T2) =:= {ok, v},
     fa_jobberteke(T1) =:= {ok, 7},
     fa_jobberteke(T2) =:= {ok, y},
     fa_jobberteke(level) =:= error,
     rendezett_fa(T1) =:= true,
     rendezett_fa(T2) =:= false,
     rendezett_fa(T3) =:= false,
     rendezett_fa_2(T1) =:= true,
     rendezett_fa_2(T2) =:= false,
     rendezett_fa_2(T3) =:= false,
     utak(T1) =:= [{4,[]},{3,[4]},{6,[4]},{5,[4,6]},{7,[4,6]}],
     utak(T2) =:= [{a,[]},{b,[a]},{v,[a,b]},{c,[a]},{d,[a,c]},{w,[a,c,d]},
                   {x,[a,c,d,w]},{f,[a,c,d]},{x,[a,c,d,f]},{y,[a,c,d,f]}],
     cutak(x,T1) =:= [],
     cutak(x,T2) =:= [{x,[a,c,d,w]},{x,[a,c,d,f]}],
     cutak_2(x,T1) =:= cutak(x,T1),
     cutak_2(x,T2) =:= cutak(x,T2)
    ]
;
test(lusta) ->
    [
     nth(infseq(1),99) =:= 99,
     nth_2(infseq(1),99) =:= {ok, 99},
     nth_2(nil, 5) =:= error,
     nth(fibs(0,1), 100) =:= 218922995834555169026,
     nth_2(fibs(0,1), 100) =:= {ok,218922995834555169026},
     nth_2(fibs(0,1), -1) =:= error,
     fib(99) =:= 218922995834555169026
    ]
;
test(nzh) ->
    [
     szigetek([]) =:= [],
     szigetek([-1,0,-2,-3,0,0]) =:= [],
     szigetek([1,2,3,0,0,4,5,6]) =:= [[1,2,3],[4,5,6]],
     szigetek_2([]) =:= [],
     szigetek_2([-1,0,-2,-3,0,0]) =:= [],
     szigetek_2([1,2,3,0,0,4,5,6]) =:= [[1,2,3],[4,5,6]],
     szfold([0,1,0,3,4,0,0,1]) =:= [{1,1}, {2,4}, {1,1}],
     szfold([0,0,30,4,10,0,0,100]) =:= [{3,30}, {1,100}],
     szfold([0,0,0]) =:= [],
     szfold_2([0,1,0,3,4,0,0,1]) =:= [{1,1}, {2,4}, {1,1}],
     szfold_2([0,0,30,4,10,0,0,100]) =:= [{3,30}, {1,100}],
     szfold_2([0,0,0]) =:= [],
     dec2hex(0) =:= "0",
     dec2hex(7) =:= "7",
     dec2hex(14) =:= "E",
     dec2hex(75) =:= "4B",
     d2hfa(e) =:= e,
     d2hfa({n, 14, e, e}) =:= {n, "E", e, e},
     d2hfa({n, 75, {n,200,e,e}, {n,35,e,e}}) =:=
         {n, "4B", {n,"C8",e,e}, {n,"23",e,e}}
    ]
.
```