

Deklaratív programozás, 6. gyakorlat (Erlang)
2018.11.13. -- 2018.11.15.
Néhány feladathoz segítséget talál a feladatsor végén.

I. RÉSZ: LISTA

1. Mátrix transzponáltja

```
-spec transpose(M::list([any()])) -> MT::list([any()]).  
%% M transzponáltja MT.
```

```
transpose([[a,b], [c,d], [e,f]]) := [[a,c,e], [b,d,f]].
```

II. RÉSZ: BINÁRIS FA

A példasorban a `fa()` és `egeszfa()` adattípusokat a következő módon definiáljuk:

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

Tehát egy `'fa()'` típusú Erlang-kifejezés

- vagy egy olyan adatot tartalmazó csomópont lehet, amely további két `'fa()'` típusú értéket tartalmaz; az első a bal részfa, a második a jobb részfa, az adatot pedig címkének nevezzük;
- vagy címke nélküli levél,

Egy `'egeszfa()'` olyan `'fa()'`, amelynek minden címkéje egész.

A példákban felhasznált változók értéke:

```
T1 = {4,  
      {3,level,level},  
      {6,  
       {5,level,level},  
       {7,level,level}}},  
T2 = {a,  
      {b, {x,level,level}, level},  
      {c,  
       level,  
       {d,  
        {x,{e,level,level},level},  
        {a, {x,level,level},{x,level,level}}}}}
```

2. Címkék felsorolása hatékonyan

```
-spec cimkek(F::fa()) -> L::[any()].  
%% L az F címkéinek listája inorder sorrendben. (Tipp a végén.)
```

```
cimkek(T1) := [3,4,5,6,7].
```

3. Bináris fa bal szélső címkéjének meghatározása

```
-spec fa_balerteke(F::fa()) -> {ok, C::any()} | error.  
%% Egy nemüres F fa bal oldali szélső címkéje C; rá és minden  
%% felmenőjére igaz, hogy bal oldali gyermek. Üres fa esetén  
%% 'error' a válasz.
```

```
fa_balerteke(T1) := {ok, 3}.  
fa_balerteke(level) := error.
```

4. Bináris fa jobb szélső címkéjének meghatározása

```
-spec fa_jobberteke(F::fa()) -> {ok, C::any()} | error.  
%% Egy nemüres F fa jobb oldali szélső címkéje C; rá és minden  
%% felmenőjére igaz, hogy jobb oldali gyermek. Üres fa esetén  
%% 'error' a válasz.
```

```
fa_jobberteke(T1) := {ok, 7}.  
fa_jobberteke(level) := error.
```

5. Bináris fa rendezettség

Egy bináris fa rendezett, ha inorder bejárásakor a címkéi szigorúan monoton növekednek, azaz a csomópontjai kielégítik a keresőfa-tulajdonságot: minden egyes csomópont címkéje nagyobb a bal oldali gyermekei címkéjénél és kisebb a jobb oldali gyermekei címkéjénél.

Oldja meg kétféleképpen, az alábbi segédfüggvények felhasználásával:

- a 2. feladat szerinti `cimkek/1` és a `lists:sort/1` függvényekkel;
- a 3. és 4. feladatok szerinti `fa_balerteke/1` és `fa_jobberteke/1` függvényekkel.

```
-spec rendezett_fa(F::fa()) -> B::boolean().  
%% B igaz, ha az F fa rendezett.
```

```
rendezett_fa(T1) := true.  
rendezett_fa(T2) := false.
```

6. Bináris fa összes címkéjének útvonala

Egy adott csomópont útvonalának nevezzük azon csomópontok címkéinek listáját, amelyek az adott csomópontig el lehet jutni.

```
-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 csomópontig vezető  
%% útvonal. (Tipp a végén.)
```

```
utak(T1) := [{4, []}, {3, [4]}, {6, [4]}, {5, [4,6]}, {7, [4,6]}].  
utak(T2) := [{a, []},  
             {b, [a]},  
             {x, [a,b]},  
             {c, [a]},  
             {d, [a,c]},  
             {x, [a,c,d]},  
             {e, [a,c,d,x]},  
             {a, [a,c,d]},  
             {x, [a,c,d,a]},  
             {x, [a,c,d,a]}].
```

7. Címke összes előfordulása bináris fában útvonallal

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

- oldja meg listanézetrel és az `utak/1` felhasználásával,
- oldja meg memóriatakarékosabban úgy, hogy csak a keresett útvonalakat tárolja az összes útvonal helyett. (Tipp a végén.)

```
cutak(x, T1) := [].  
cutak(x, T2) := [{x, [a,b]}, {x, [a,c,d]}, {x, [a,c,d,a]}, {x, [a,c,d,a]}].
```

III. RÉSZ: LUSTA LISTA

8. Lusta lista n-edik eleme

- spec `nth(L::lazy:list(), N::integer()) -> X::any()`.
%% Az L lusta lista N-edik eleme X (számozás 1-től).
- spec `nth2(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 $N < 1$.

SEGÍTSÉG A MEGOLDÁSHOZ

2. Cél a lineáris időigényű algoritmus. Javasolt segédfüggvény:

```
-spec cimkek(F::fa(), L1::[any()]) -> L::[any()].
%% L az F címkéinek listája inorder sorrendben L1 elé fűzve.
```

6. Bináris fa összes címkéjének útvonala

Javasolt segédfüggvény:

```
-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 útvonal elé fűzve.
```

7. Címke összes előfordulása bináris fában útvonallal

ad b) A megoldás nagyon hasonló az 6. feladat megoldásához (vö. utak/2 segédfüggvény), de a fa gyökerének címkéjét csak feltételelesen tárolja el.

A következő feladat példa arra, hogy a Deklaratív programozás nagyzárthelyi Erlang részében milyen "mitírki" jellegű feladatokat kell megoldani.

"Mitírki" jellegű NZH-mintafeladat Erlangból
=====

9. Mi az X változó értéke? A deklarációk függetlenek egymástól; az is_space/1 karaktervizsgáló függvény modulnév megadása nélkül hívható.

```
is_space(X) -> X =:= 32.
```

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

A következő feladatok bemutatják, hogy a nagyzárthelyi Erlang részében milyen jellegű programozási feladatokat kell megoldani, milyen szerkezetben.

Két, egymásra épülő feladatot kell kidolgozni: először egy segédfüggvényt, majd ezt felhasználva egy, a teljes feladat megoldó függvényt kell írni.

Programozási NZH-mintafeladatok (M1 és M2) Erlangból
=====

10. Az M1 feladat segéd eljárása

írjon szigetek néven olyan függvényt, amely visszaadja egy lista pozitív elemekből álló, folytonos, semelyik irányba ki nem terjeszthető részlistáinak listáját. Törekedjék hatékony megoldásra!

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

Példák:

```
szigetek([]) =:= [].
szigetek([-1,0,-2,-3,0,0]) =:= [].
szigetek([1,2,3,0,0,4,5,6]) =:= [[1,2,3],[4,5,6]].
szigetek([2,1,3,0,0,7,6,-1,-3,0,7,5,6]) =:= [[2,1,3],[7,6],[7,5,6]].
```

11. A teljes M1 feladat

írjon szfold néven olyan függvényt a szigetek/1 függvény felhasználásával, amely egy negatív számokat nem tartalmazó egészlista szárazföldjeinek hosszából és legmagasabb pontjából álló párok listáját adja eredményül! Szárazföldnek a csupa pozitív számból álló, legalább egyelemű, maximális hosszúságú, folytonos részlistát nevezzük.

```
-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.
```

Példák:

```
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]) =:= [].
```

12. Az M2 feladat segéd eljárása

írjon dec2hex néven olyan függvényt, amely kiszámolja egy decimális számnak a 0--9 és A--F karakterek füzéréként (sztring) ábrázolt hexadecimális alakját. A fűzér utolsó karaktere legyen a legkisebb, egyes helyiértékű számjegy. Ügyeljen a 0 helyes kezelésére!

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

Példák:

```
dec2hex(0) =:= "0".
dec2hex(7) =:= "7".
dec2hex(14) =:= "E".
dec2hex(75) =:= "4B".
```

13. A teljes M2 feladat

írjon d2hfa néven olyan függvényt a dec2hex/1 függvény felhasználásával, amely a

```
-type fa(Elem) :: e | {n, Elem, fa(Elem), fa(Elem)}
```

típusspecifikációval megadott, fa(integer()) típusú bemenő argumentumát fa(string()) típusúvá alakítja úgy, hogy minden 'n' csomópontban található egész számot lecserél a fűzéréként ábrázolt hexadecimális megfelelőjére. További segédfüggvényeket ne definiáljon!

```
-spec d2hfa(Fa0::fa(integer())) -> Fa::fa(string()).
%% A Fa0 decimális számokat tartalmazó fa hexadecimális számokat
%% tartalmazó megfelelője Fa.
```

Példák:

```
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}}.
```

----- \$LastChangedDate: 2018-11-16 16:29:27 +0100 (p, 16 nov 2018) \$ -----