Deklaratív programozás, 4. gyakorlat (Erlang)
2018.10.16. -- 2018.10.18.
Néhány feladathoz segítséget talál a feladatsor végén.

I. RÉSZ: LISTAKEZELÉS

---

1. Számsorozat jobbrekurzívan (vö lists:seq/2)

```
-spec seq(F::integer(), T::integer()) -> S::[integer()].
%% S = [F, F+1, ..., T].

seq(10,13) =:= [10,11,12,13].
```

---

2. Listaelemek különböző voltának vizsgálata

```
-spec all_different(Xs::[any()]) -> B::boolean().
%% B igaz, ha az L listában csupa különböző értékű elem van.

all_different([1,2,3,1]) =:= false.
all_different([1,2,3]) =:= true.
```

A megoldást valósítsa meg többféleképpen is:
a) lists:member felhasználásával (a rekurzív hívás lusta kiértékelésű legyen);
b) lists:usort felhasználásával.

---

3. Listának legalább két eleme van - hatékonyan, a length/1 függvény nélkül!

```
-spec van2eleme(L::list()) -> R::boolean().
%% R =:= length(L) >= 2.
```

---

4. Lista ismétlődő elemei -- használjuk fel a lists:zip/2 függvényt!

```
-spec duplak(Xs::[any()]) -> Ys::[any()].
%% Ys az Xs azon elemeinek listája, amelyek az őket követő elemmel egyelőek.

duplak([1,2,3]) =:= [].
duplak([1,1,2,3,3,3]) =:= [1,3,3].
```

---

5. Lista minden elemének ellenőrzése (vö lists:all/2)

```
-spec all(Pred::fun((T::any()) -> boolean()), L::[T::any()]) -> B::boolean().
%% B akkor és csak akkor igaz, ha Pred teljesül az L minden elemére.

all(fun erlang:is_atom/1,[a,b]) and not all(fun erlang:is_atom/1,[a,1]).
```

A megoldást valósítsa meg többféleképpen is. Melyik hatékonyabb és miért?
a) lists:map és lists:foldl felhasználásával;
b) rekurzívan, ügyelve a rekurzív hívás lusta kiértékelésére.

---

6. Platók hossza

```
-spec platok_hossza(Xs::[any()]) -> PHs::[{P::any, H::integer()}].
%% Platónak nevezzük az egyenlő értékű elemek sorozatát egy listában.
%% PHs olyan {P, H} párok listája, amelyekben P a platót képező
%% érték, H pedig e plató hossza (= azonos értékű elemeinek száma).

platok_hossza([a,a,a,b,b,d,f,f,h]) =:= [{a,3},{b,2},{d,1},{f,2},{h,1}].
```

---

7. Lista részlistája listanézettel, a lists:nth/2 és lists:seq/2 felhasználásával.

```
-spec sublist(L0::[any()], S::integer(), N::integer()) -> L::[any()].
%% Az L0 lista S-edik elemétől kezdődő és N hosszú részlistája az L lista.

sublist([a,b,c], 2, 1) =:= [b].
sublist([a,b,c], 2, 2) =:= [b,c].
```

---

8. Mátrix részmátrixa

```
-spec kozepe(M::list([any()])) -> M1::list([any()]).
%% M1 az M n*n-es négyzetes mátrix olyan (n/2)*(n/2) méretű részmátrixa,
%% mely az n/4+1. sor n/4+1. oszlopának elemétől kezdődik.

M=[[a,b,e,f],
   [c,d,g,h],
   [i,j,m,n],
   [k,l,o,p]].

kozepe(M) =:= [[d,g],[j,m]].
```

A megoldást valósítsa meg többféleképpen is:
a) Használja fel listanézetben a lists:sublist/3 függvényt:
   -spec sublist(L1::list(), Start::integer(), Len::integer()) -> L2::list().
b) Használja fel listanézetben a lists:nth/2 függvényt:
   -spec nth(N::integer(), List::list()) -> Elem::any().

---

9. Mátrix középső elemei

```
-spec laposkozepe(M::lista([any()])) -> L::[any()].
%% Az L lista az M mátrix közepe elemeinek listája.

laposkozepe(M) =:= [d,g,j,m].
```

A megoldást valósítsa meg többféleképpen is:
a) lists:flatten/1 és kozepe/1 felhasználásával.
   % flatten(DeepLst) = DeepLst beágyazott elemeinek kilapított listája.
   % Pl.: lists:flatten([1,[2,3],[[[[4]]],[5,[6]]]]) =:= [1,2,3,4,5,6].
b) Használja fel listanézetben a lists:nth/2 függvényt.

---

10. Részmátrix sor és oszlop elhagyásával

```
-spec pivot(M::list([any()]), R::integer(), C::integer())-> M1::list([any()]).
%% M1 az M mátrix R-edik sorának és C-edik oszlopának elhagyásával keletkezik.

pivot(M,2,3) =:= [[a,b,f],[i,j,n],[k,l,p]].
```

A megoldáshoz használja fel listanézetben a lists:nth/2 függvényt.

---

11. 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]].
```

TOVÁBBI GYAKORLÓ FELADATOK OTTHONRA

+1. Öszetett számok

```
-spec osszetett(K::integer()) -> L::[integer()].
%% L tartalmazza az összetett számokat 4..K*K között, ismétlődés lehet.
"Eratoszthenész szitája": bejárjuk minden szám többszöröseit, és
megjelöljük őket összetettként. Felhasználható lists:seq/3 függvény:
%% seq(F, T, D) =:= [F, F+D, F+2D, ..., F+KD], ahol F+KD =< T < F+(K+1)D.
osszetett(5) =:= [4,6,8,10,12,14,16,18,20,22,24,
                  6,9,12,15,18,21,24,    8,12,16,20,24,    10,15,20,25].
```

+2. Prímszámok

```
-spec primek(K::integer()) -> L::[integer()].
%% L tartalmazza a prímszámokat 2..K*K között.
primek(5) =:= [2,3,5,7,11,13,17,19,23].
```

+3. Listák cipzárazása (vö lists:zip/2, lists:unzip/1)

```
-spec zip(Xs::[any()], Ys::[any()]) -> XYs::[{any(), any()}].
-spec unzip(XYs::[{any(), any()}]) -> {Xs::[any()], Ys::[any()]}.
%% XYs olyan párok listája, amelynek első eleme az Xs, második eleme
%% az Ys lista azonos pozíciójú eleme.
zip([1,2,3], [a,b,c]) =:= [{1,a},{2,b},{3,c}].
unzip([{1,a},{2,b},{3,c}]) =:= {[1,2,3], [a,b,c]}.
```

II. RÉSZ: BINÁRIS FÁK

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

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

---

12. Bináris egészfa minden elemének növelése

```
    -spec fa_noveltje(F0::egeszfa()) -> F::egeszfa().
    %% Az F fa minden címkéje eggyel nagyobb az F0 azonos helyen lévő címkéjénél.

    fa_noveltje(T1) =:= {5,{4,level,level},{7,{6,level,level},{8,level,level}}}.
```

---

13. Bináris fa tükörképe

```
    -spec fa_tukorkepe(F0::fa()) -> F::fa().
    %% F az F0 fa tükörképe.

    fa_tukorkepe(T1) =:= {4,{6,{7,level,level},{5,level,level}},{3,level,level}}.
```

---

14. Bináris fa inorder bajárása

```
    -spec inorder(F::fa()) -> L::list().
    %% L az F fa elemeinek a fa inorder bejárásával létrejövő listája.
```

---

15. Címke előfordulása (rendezetlen) bináris fában

```
    -spec tartalmaz(C::any(), F::fa()) -> B::boolean().
    %% B igaz, ha C az F fa valamely címkéje.
    tartalmaz(x, T1) =:= false.
    tartalmaz(x, T2) =:= true.
```

---

16. Címke összes előfordulásának száma bináris fában

```
    -spec elofordul(C::any(), F::fa()) -> N::integer().
    %% A C címke az F fában N-szer fordul elő.

    elofordul(x, T1) =:= 0.
    elofordul(x, T2) =:= 4.
```

---

17. Címkék felsorolása hatékonyan

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

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

SEGÍTSÉG A MEGOLDÁSHOZ

4. Lista ismétlődő elemei
   A lista helyett tekintsük párok listáját. Cipzárazzuk össze a lista első, illetve utolsó elemének elhagyásával keletkező listákat, például az [1,1,2,3,3,3] esetén képezzük a [1,1,2,3,3], [1,2,3,3,3] listák cipzárasával keletkező listát: [{1,1},{1,2},{2,3},{3,3},{3,3}].

```
   -spec sublist(L::[any()], N::integer()) -> L2::[any()].
   %% L2 az L első N eleméből álló részlistája.
```

17. 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.
```

----- $LastChangedDate: 2018-10-18 12:58:28 +0200 (cs, 18 okt 2018) $ ------