Deklaratív programozás, 6. gyakorlat
Erlang programozás: listák, fastruktúrák
2014. 11. 06.

Megjegyzés: néhány feladathoz segítséget talál a feladatsor végén.

I. RÉSZ: LISTAKEZELÉS

---

1. Számsorozat jobbrekurzívan (lists:seq/2)

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

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

---

2. Mátrix részmátrixa

```
%% @spec kozepe(M::[[term()]]) -> M1::[[term()]].
%% 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(), S::int(), L::int()) -> L2::list().`
- b) Használja fel listanézetben az lists:nth/2 függvényt:
  `%% @spec nth(N::int(), List::list()) -> Elem::term().`

---

3. Mátrix középső elemei

```
%% @spec laposkozepe((M::[[term()]]) -> L::[term()].
%% 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 az lists:nth/2 függvényt.

---

4. Részmátrix sor és oszlop elhagyásával

```
%% @spec pivot((M::[[term()]], R::int(), C::int()) -> M1::[[term()]].
%% M1 az M mátrix R. sorának és C. oszlopának elhagyásával keletkezik.

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

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

---

5. Lista elemeinek egyezése

```
%% @spec all_different(Xs::[any()]) -> B::bool().
%% B igaz, ha az L lista minden eleme különbözik.

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

A megoldást valósítsa meg többféleképpen is:
- a) lists:member felhasználásával,
  ügyelve a rekurzív hívás lusta kiértékelésére;
- b) lists:usort felhasználásával.

---

6. Listák cipzárazása (lists:zip/2, lists:unzip/1)

```
%% @spec zip(Xs::[term()], Ys::[term()]) -> XYs::[{term(), term()}].
%% @spec unzip(XYs::[{term(), term()}]) -> {Xs::[term()], Ys::[term()]}.
%% 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]}.
```

---

7. Lista ismétlődő elemei – használjuk fel a zip függvényt!

```
%% @spec duplak(Xs::[term()]) -> Ys::[term()].
%% Ys az Xs lista azon elemei, melyek azonosak az őket követő elemmel.

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

---

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

```
%% @spec all(P::fun(T) -> boolean(), L::[T]) -> boolean().
%% Igaz, ha L minden elemére P igaz, különben hamis.

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.

---

9. Mátrix transzponáltja

```
%% @spec transpose(M::[[term()]]) -> MT::[[term()]].
%% M transzponáltja MT.

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

TOVÁBBI GYAKORLÓ FELADATOK OTTHONRA

+1. Összetett 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].
```

II. RÉSZ: BINÁRIS FÁK

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

```
% @type fa()      = level | {term(),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}}}}}.
```

---

10. Egészfa minden elemének növelése

```
    %% @spec fa_noveltje(F0::egeszfa()) -> F::egeszfa().
    %% Az F fa az F0 egészfának olyan másolata, amelynek ugyanannyi levele és
    %% csomópontja van, mint az F0-nak, ám F minden címkéje pontosan eggyel
    %% nagyobb, mint az F0 megfelelő címkéje.

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

---

11. Bináris fa tükörképe

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

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

---

12. Bináris fa legszélső címkéjének meghatározása

```
    a) %% @spec fa_balerteke(F::fa()) -> {ok, C::term()} | error.
       %% A nemüres F fa bal oldali szélső címkéje C, amelyre és minden
       %% felmenőjére igaz, hogy bal oldali gyermek; üres fa esetén 'error'.
    b) %% @spec fa_jobberteke(F::fa()) -> {ok, C::term()} | error.
       %% A nemüres F fa jobb oldali szélső címkéje C; üres fa esetén 'error'.

    fa_balerteke(T1) =:= {ok, 3}.
    fa_balerteke(level) =:= error.
    fa_jobberteke(T1) =:= {ok, 7}.
```

---

13. Bináris fa rendezettsége

```
    Egy bináris fa rendezett, ha inorder bejárásakor a címkéi szigorúan monoton
    növekednek, azaz a csomópontjai kielégíti a keresőfa-tulajdonságot: minden
    egyes csomópont címkéje nagyobb a bal oldali gyermekei címkéinél és kisebb
    a jobb oldali gyermekei címkéinél. (Tipp a végén.)

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

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

---

14. Címke előfordulása (rendezetlen) bináris fában

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

---

15. Címke összes előfordulásának száma bináris fában

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

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

---

16. Bináris fa összes címkéjének útvonala

```
    Egy adott csomópont útvonalának nevezzük azon csomópontok címkéinek
    listáját, amelyeken át a fa gyökerétől az adott csomópontig el lehet jutni.

    %% @type ut() = [term()].
    %% @spec utak(F::fa()) -> CimkezettUtak::[{term(), ut()}].
    %% A CimkezettUtak lista az F fa minden csomópontjához egy kételemű ennest
    %% társít, amelynek első eleme a csp. címkéje, második eleme a csp.
    %% útvonala. (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]}].
```

---

17. Címke összes előfordulása bináris fában útvonallal

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

    a) oldja meg listanézettel és az utak/1 felhasználásával,
    b) 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]}].
```

---

18. Címkék felsorolása hatékonyan

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

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

SEGÍTSÉG A MEGOLDÁSHOZ

7. 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::[term()], N::integer()) -> L2::[term()].
   %% L2 az L első N eleméből álló részlistája.
13. Bináris fa rendezettsége
   A megoldásban célszerű a 3.a) és 3.b) feladatok megoldásait segédeljárásként felhasználni, így nem szükséges további segédeljárást definiálni.
16. Bináris fa összes címkéjének útvonala
   Javasolt segédeljárás:
   %% @spec utak(F::fa(),Eddigi::ut())->CimkezettUtak::[{C::term(),U::ut()}].
   %% A CimkezettUtak lista az F fa minden csomópontjához egy kételemű ennest
   %% társít, amelynek első eleme (C) a csp. címkéje, második eleme (U) az
   %% Eddigi útvonal és a csp. útvonala összefűzve.
17. Címke összes előfordulása bináris fában útvonallal
   b) A megoldás nagyon hasonló a 7. megoldáshoz, de a fa gyökerének címkéjét csak feltételesen tároljuk el.
18. Cél a lineáris időigényű algoritmus. Javasolt segédfüggvény:
   %% @spec cimkek(F::fa(), L1::[term()]) -> L::[term()].
   %% L az F címkéinek listája inorder sorrendben L1 elé fűzve.

----- $LastChangedDate: 2014-11-02 15:51:11 +0100 (Sun, 02 Nov 2014) $ ------
```
```