Deklaratív programozás, 4. gyakorlat
Erlang programozás
2016.10.13.

Megjegyzés: 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. Listák cipzárazása (vö 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]}.
```

---

3. Lista kilapítása (lists:flatten/1)

```
%% @spec flatten(DeepList::list()) -> L::list().
%% A DeepList - tetszőleges mélységű beágyazott listákban tartalmazott -
%% elemeit az L listában "kibontva" tartalmazza úgy, hogy az L listában
%% már nem szerepel elemként további lista.

flatten([1,[2,3],[[[[4]]],[5,[6]]]]) =:= [1,2,3,4,5,6].

Megjegyzés: naív változathoz használható a lists:append/2 (++).
* Szorgalmi: append nélkül, flatten/2 segédfüggvény bevezetésével.
```

---

4. 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().
```

---

5. 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.
```

---

6. 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.
```

---

7. 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.
```

---

8. Listában van legalább 2 elem - hatékonyan, a length/1 függvény nélkül!

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

[van2eleme(L) || L<-[[],[a],[a,b],[a,b,c]]] =:= [false,false,true,true].
```

---

9. Lista ismétlődő elemei - használjuk fel a lists:zip/2 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].
```

---

10. 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.
```

---

11. Mátrix transzponáltja

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

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

SEGÍTSÉG A MEGOLDÁSHOZ

9. 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.
```

II. RÉSZ: 8 VEZÉR A SAKKTÁBLÁN

Feladatok otthoni gyakorláshoz

Feladat: 8 vezér (királynő) elhelyezése a sakktáblán úgy, hogy ne üssék egymást. Az alábbi feladatok egy lehetséges megoldás felépítéséhez vezetnek.

```
%% @type sor()    = 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8.
%% @type oszlop() = 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8.  % abcdefgh helyett.
%% @type jelolt() = [oszlop()].
```

Mivel egy megoldásban bármely sorban vezér csak a többitől különböző oszlopban lehet, ezért a megoldások az oszlopindexek permutációival leírhatók. Vagyis egy megoldás egy olyan lista, amelynek ha az i. eleme j, akkor a sakktábla i. sorában a j. oszlopban van vezér. Például jobbra a [4,7,3,8,2,5,1,6] megoldás látható. Ezzel automatikusan kizárjuk, hogy két vezér azonos oszlopban vagy sorban legyen, csak azt kell vizsgálni, hogy átlósan ütésben vannak-e.

```
. . . X . . . .
. . . . . . X .
. . X . . . . .
. . . . . . . X
. X . . . . . .
. . . . X . . .
X . . . . . . .
. . . . . X . .
(1 2 3 4 5 6 7 8)
(a b c d e f g h)
```

---

12. Számpárok összeg-különbség párjainak előállítása (koordináta-transzformáció)

```
%% @spec atlok([{R::sor(), C::oszlop()}]) ->
%%           [Atlok::{integer(), integer()}].
%% Minden {R,C} párhoz egy Atlok számpárt rendel, amelynek első eleme
%% R+C, második eleme R-C.
%% Ezzel egy {sor(),oszlop()} koordinátához rendelhető, hogy "hányadik"
%% átlóban van: az összegnél balról, a különbségnél jobbról számítva.
%% Egy 3x3-as táblán az  R+C értékek:  R-C értékek:  Átlók:
%%                       2 3 4         0 -1 -2       {2,0} {3,-1} {4,-2}
%%                       3 4 5         1  0 -1       {3,1} {4,0}  {5,-1}
%%                       4 5 6         2  1  0       {4,2} {5,1}  {6,0}

atlok([{1,1}, {2,3}]) =:= [{2,0}, {5,-1}].
```

---

13. Ütések ellenőrzése

```
%% @spec nem_utik_egymast(Js::jelolt()) -> B::bool().
%% B igaz, ha a vezérek Js-beli elhelyezése helyes, vagyis nem ütik
%% egymást egy length(Js) sorból és lists:max(Js) oszlopból álló táblán.
Felhasználandó: all_different/1, seq/2, zip/2, atlok/1, unzip/1.

nem_utik_egymast([1,3,5]) =:= true.
nem_utik_egymast([1,5,3]) =:= false.
```

---

14. 1..8 számok összes permutációjának felsorolása

```
%% @spec perms() -> Ps::[jelolt()].
%% Ps az 1..8 számok összes lehetséges permutációját tartalmazó lista.

perms() =:= [[1,2,3,4,5,6,7,8], [1,2,3,4,5,6,8,7] | ... ].
```

---

15. 8 vezér -- kimerítő keresés (generate and test)

Oldjuk meg a feladatot kimerítő kereséssel: először generáljuk az összes permutációt, majd kiszűrjük azokat, amelyek helyes megfejtések.
Felhasználandó: listanézet, perms/0, nem_utik_egymast/1.

```
%% @spec vezerek8_1() -> [jelolt()].

vezerek8_1() =:= [ [1,5,8,6,3,7,2,4] | ... ].
length(vezerek8_1()) =:= 92.
```

---

16. 8 vezér -- hatékonyabb megoldás

Permutációk generálása közben is végzünk szűrést a részmegoldásra: ha már K=<8 sorhoz rendeltünk oszlopot, a K méretű részmegoldást is ellenőrizhetjük a nem_utik_egymast/1 függvénnyel.
Nem használható fel a perms/0 függvény, hiszen a generátorok közé kell a szűréseket betenni.

```
_ = statistics(runtime),           % idő lekérdezése
V1 = vezerek8_1(),                 % V1 az összes megoldás
{_,Time1} = statistics(runtime),   % utolsó lekérdezés óta eltelt idő
V2 = vezerek8_2(),                 % V2 az összes megoldás
{_,Time2} = statistics(runtime),   % utolsó lekérdezés óta eltelt idő
lists:sort(V1) =:= lists:sort(V2)  % sorrendet nem kötjük meg
    andalso Time2 < Time1.         % de bizonyosan gyorsabb
```

---

17. A sakktábla mérete 8 helyett N (paraméter). Keressük a megoldásokat kimerítő kereséssel az 1..N számok permutációi között.

---

18. N méretű sakktábla hatékonyabb megoldása: a részmegoldásokat is ellenőrizzük.

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].
```

----- $LastChangedDate: 2016-10-11 15:56:32 +0200 (k, 11 okt 2016) $ -------