Deklaratív programozás, 6. gyakorlat
Erlang programozás: fastruktúrák, listák
2013. 05. 15.

Megjegyzés: néhány feladathoz segítséget talál a feladatsor végén.

I. 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}}}}}.
```

---

1. 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}}}.
```

---

2. 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}}.
```

---

3. 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}.
```

---

4. 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.
```

---

5. 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.
```

---

6. 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.
```

---

7. 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]}].
```

---

8. 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]}].
```

II. RÉSZ: LISTAKEZELÉS

---

9. 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]}.
```

---

10. Lista ismétlődő elmei – 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].
```

---

11. 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.
```

---

12. 1..6 számok összes ismétléses variációinak felsorolása

```
%% @spec descs() -> Ps::[[integer()]].
%% Ps az 1..6 számok összes ismétléses variációját tartalmazó lista.

descs() =:= [[1,1,1,1,1,1], [1,1,1,1,1,2], [1,1,1,1,1,3], [1,1,1,1,1,4],
             [1,1,1,1,2,1], [1,1,1,1,2,2] | ... ].
```

---

13. 1..6 számok összes permutációjának felsorolása

```
%% @spec perms() -> Ps::[[integer()]].
%% Ps az 1..6 számok összes lehetséges permutációját tartalmazó lista.

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

SEGÍTSÉG A MEGOLDÁSHOZ

---

4. Bináris fa rendezettsége
   A megoldásban célszerű a 3.a) és 3.b) feladatok megoldásait segédeljárásként felhasználni, így nem szükséges további segédeljárást definiálni.

---

7. 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.
```

---

8. Címke összes előfordulása bináris fában útvonallal
   b) A megoldás nagyon hasonló a 7. megoldáshoz, de a fa gyökerének címkéjét csak feltételesen tároljuk el.

---

10. Lista ismétlődő elmei
    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.
```

---

11. Lista elemeinek egyezése

```
% @spec lists:member(E::term(), L::[term()]) -> R::bool().
% R igaz, ha E eleme L-nek.

Lusta kiértékelésű: 'andalso'/2, case...of.  Mohó: 'and'/2.

% @spec lists:usort(L1::[term()]) -> L2::[term()].
% L2 az L1 lista elemeinek ismétlés nélküli (unique) rendezett listája.
Például: lists:usort([2,1,3,1]) =:= [1,2,3].
```

---

12. 1..6 számok ismétléses variációinak felsorolása
    A legrövidebb megoldáshoz használjunk listanézetet.

---

13. 1..6 számok összes permutációjának felsorolása
    A legrövidebb megoldáshoz megszűrhetjük az ismétléses variációkat az all_different segítségével.

    Nagyságrendileg gyorsabb megoldás is adható egy ügyesebb listanézettel. Emlékeztetőül: A--B az A lista azon elemei, melyek nem elemei a B listának, ha all_different(A) és all_different(B).

----- $LastChangedDate: 2013-05-03 11:27:46 +0200 (Fri, 03 May 2013) $ ------