Deklaratív programozás, 3. gyakorlat
Erlang programozás: struktúrák, listák
2010. 03. 17.

Az alábbi példasorban a ‘fa()’ adattípust a következõ módon definiáljuk:

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

Tehát egy ‘fa()’ típusú Erlang-kifejezés
- vagy egy egész számból álló levél,
- vagy egy olyan csomópont lehet, amely két ‘fa()’ típusú értéket tartalmazó párból áll.

Az egészlista olyan lista, amelynek elemei egészek:

```
%% @type egeszlista() = [integer()].
```

Írjon olyan Erlang-eljárást, amely megfelel az adott fejkommentnek. Az adott feladat megoldásához felhasználhat korábbi sorszámú feladatokban definiált eljárásokat. Ha ilyeneken kívül is szükséges segédeljárás definiálása, azt külön jelezzük.

---

1. Bináris fa csomópontjainak száma

```
%% @spec faPontszama(F::fa()) -> N::integer().
%% Az F fa csomópontjainak száma N.

faPontszama({1,{2,3}}) == 2.
faPontszama({1,{2,{3,4}}}) == 3.
```

2. Bináris fa mélysége

Egy fa mélységén az egymásba skatulyázott csomópontjainak maximális számát értjük.

```
%% @spec faMelysege(F::fa()) -> M::integer().
%% Az F fa mélysége M.

Tipp: az érthetõséget növelheti a max függvény használata.

faMelysege({{1,4},{2,3}}) == 2.
faMelysege({1,{2,{4,3}}}) == 3.
```

3. Lista hossza

Egy lista elemeinek számát a lista hosszának nevezzük.

```
%% @spec listaHossza(L::egeszlista()) -> H::integer().
%% Az L egészlista hossza H.

listaHossza([1,3,5]) == 3.
```

4. Bináris fa minden levélértékének növelése

```
%% @spec faNoveltje(F0::fa()) -> F::fa().
%% Az F fa az F0 fának olyan másolata, amelynek ugyanannyi levele
%% és csomópontja van, mint az F0-nak, ám az F minden levélének értéke
%% pontosan eggyel nagyobb, mint az F0 megfelelõ levelének az értéke.

faNoveltje({1,{2,3}}) == {2,{3,4}}.
```

5. Számlista minden elemének növelése

```
%% @spec listaNoveltje(L0::egeszlista()) -> L::egeszlista().
%% Az L egészlista az L0 egészlistának olyan másolata, amelynek ugyanannyi
%% eleme van, mint az L0-nak, de az L minden elemének értéke pontosan
%% eggyel nagyobb, mint az L0 megfelelõ elemének az értéke.

listaNoveltje([1,5,2]) == [2,6,3].
```

6. Bináris fa legbaloldalibb levélértékének meghatározása

```
%% @spec faBalerteke(F::fa()) -> E::level().
%% Az F fa legbaloldalibb levelének értéke E.

 faBalerteke({{1,4},{2,3}}) == 1.
faBalerteke({{{9,8},7},{6,{5,4}}}) == 9.
```

7. Bináris fa legjobboldalibb levélértékének meghatározása

```
%% @spec faJobberteke(F::fa()) -> E::level().
%% Az F fa legjobboldalibb levelének értéke E.

faJobberteke({{1,4},{2,3}}) == 3.
faJobberteke({{{9,8},7},{6,{5,4}}}) == 4.
```

8. Egy lista utolsó elemének meghatározása

```
%% @spec listaUtolsoEleme(L::egeszlista()) -> E::integer().
%% Az L egészlista utolsó eleme E.

listaUtolsoEleme([5,1,2,8,7]) == 7.
```

9. Bináris fa részfái

Egy fa (nem feltétlenül valódi) részfájának nevezzük saját magát, valamint -- ha a fa egy csomópont -- a bal és a jobb oldali ágának részfáit.

```
%% @spec faReszfai(F::fa()) -> Reszfalista::[fa()].
%% Reszfalista az F fa részfáinak listája. A listában a részfák sorrendje
%% tetszõleges, de minden részfa csak egyszer fordulhat elõ.

faReszfai({1,{2,3}}) == [{1,{2,3}},1,{2,3},2,3].
faReszfai({1,{2,{3,4}}}) == [{1,{2,{3,4}}},1,{2,{3,4}},2,{3,4},3,4].
```

10. Egy lista szuffixumai

```
    Egy n-elemû L lista k-elemû szuffixumának nevezzük azt a listát, amely az
    L utolsó k elemét tartalmazza az elemek L-beli sorrendjének megtartása
    mellett (0 =< k =< n).

    %% @spec listaSzuffixumai(L::egeszlista()) -> Szuffixlista::[egeszlista()].
    %% Szuffixlista az L lista szuffixumainak listája. A listában a szuffixumok
    %% sorrendje tetszõleges, de minden szuffixum csak egyszer fordulhat elõ.

    listaSzuffixumai([1,4,2]) == [[1,4,2],[4,2],[2],[]].
```

11. Bináris fa tükörképe

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

    faTukorkepe({{1,4},{2,3}}) == {{3,2},{4,1}}.
    faTukorkepe({{1,4},{4,1}}) == {{1,4},{4,1}}.
    faTukorkepe({{1,{2,3}},4}) == {4,{{3,2},1}}.
```

12. Bináris fa tükörszimmetrikus voltának ellenõrzése

```
    %%@spec tukorszimmetrikusFa(F::fa()) -> B::bool().
    %% B igaz, ha az F fa tükörszimmetrikus.

    tukorszimmetrikusFa({{1,4},{4,1}}) == true.
    tukorszimmetrikusFa({{1,4},{2,3}}) == false.
```

13. Bináris fa leveleiben található értékek listája

```
    %% @spec faLevelertekei(F::fa()) -> Llista::[level()].
    %% Llista az F fa leveleiben található értékek listája. A listában az elemek
    %% sorrendje tetszõleges, de minden elem csak egyszer fordulhat elõ.

    faLevelertekei({1,{2,3}}) == [1,2,3].
    faLevelertekei({{1,4},{2,3}}) == [1,4,2,3].
    faLevelertekei({{1,{2,{3,4}}},5}) == [1,2,3,4,5].
```

14. Bináris fa rendezettsége

```
    Egy fát rendezettnek mondunk, ha a fában balról jobbra haladva a
    levelekben található értékek szigorúan monoton növõ sorozatot alkotnak.

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

    A megoldásban célszerû a 6. és 7. feladat eljárásait segédeljárásként
    felhasználni, így nem szükséges további segédeljárást definiálni.

    Keressen hatékonyabb megoldást is, amelyben a fastruktúrát csak egyszer
    járja be. Ehhez egy megfelelõ segédeljárás szükséges.

    rendezettFa({{1,4},{2,3}}) == false.
    rendezettFa({{1,3},{5,{6,9}}}) == true.
```

15. Fából lista

```
    %% @spec fabolLista(F::fa()) -> Ls::egeszlista().
    %% Az F fa balról jobbra haladva elõállított elemeinek listája Ls.

    fabolLista2({{1,4},{2,3}}).
    fabolLista2({{1,3},{5,{6,9}}}).
```