Deklaratív programozás, 3. gyakorlat (Elixir), 2020.09.28. -- 2020.09.30.

Az alábbi feladatok megoldására írjon olyan Elixir-függvényeket, amelyek megfelelnek a fejkommenteknek. Írjon mindegyikből több verziót. Gondolja át, melyik hatékonyabb és miért.

Használjon listajelölőt (komprehenziót), használja az Elixir és Erlang modulok függvényeit. A feladatok megoldásához használhatja a korábbi feladatokban definiált függvényeit.

A függvényeket az IEx-ben a modulnév megadásával kell meghívni (Mnév.fnév). Az alábbi példákban nem írjuk ki a modulnevet.

BINÁRIS FÁK

A feladatsorban 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 Elixir-kifejezés
- vagy egy adatot tartalmazó csomópont, amely további két fa típusú értéket tartalmaz: az első a bal, a második a jobb részfa (az adatot a továbbiakban címkének nevezzük);
- vagy egy 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, {:v,:level,:level}, :level},
 {:c,
  :level,
  {:d,
   {:w,{:x,:level,:level},:level},
   {:f, {:x,:level,:level},{:y,:level,:level}
   }
  }
 }
}
```

---

1. Bináris egészfa minden címkéjének megnövelése 1-gyel

```
@spec fa_noveltje(f0::egeszfa) :: f::egeszfa
# Az f fa minden címkéje eggyel nagyobb az f0 fa azonos helyen lévő címkéjénél

fa_noveltje(t1) === {5,{4,:level,:level},{7,{6,:level,:level},{8,:level,:level}}}
```

---

2. 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}}
```

```
 3. Bináris fa inorder, preorder és postorder bejárása

    @spec inorder(f::fa) :: ls::[any]
    # ls az f fa elemeinek a fa inorder bejárásával létrejövő listája

    @spec preorder(f::fa) :: ls::[any]
    # ls az f fa elemeinek a fa preorder bejárásával létrejövő listája

    @spec postorder(f::fa) :: ls::[any]
    # ls az f fa elemeinek a fa postorder bejárásával létrejövő listája

    inorder(t1)   === [3,4,5,6,7]
    preorder(t1)  === [4,3,6,5,7]
    postorder(t1) === [3,5,7,6,4]

-------------------------------------------------------------------------------
 4. Címke előfordulása (rendezetlen) bináris fában

    @spec tartalmaz(f::fa, c::any) :: b::boolean
    # b igaz, ha c az f fa valamely címkéje

    tartalmaz(t1, :x) === false
    tartalmaz(t2, :x) === true

-------------------------------------------------------------------------------
 5. Címke összes előfordulásának száma bináris fában

    @spec elofordul(f::fa, c::any) :: n::integer
    # A c címke az f fában n-szer fordul elő

    elofordul(t1, :x) === 0
    elofordul(t2, :x) === 2

-------------------------------------------------------------------------------
 6. Címkék felsorolása hatékonyan: írjon lineáris időigényű algoritmust!

    @spec cimkek(f::fa) :: ls::[any]
    # ls az f címkéinek listája inorder sorrendben

    cimkek(t1) === [3,4,5,6,7]

    Javasolt segédfüggvény:
    @spec cimkek(f::fa, zs::[any]) :: ls::[any]
    # ls az f címkéinek listája inorder sorrendben zs elé fűzve

-------------------------------------------------------------------------------
 7. Bináris fa bal szélső címkéjének meghatározása

    @spec fa_balerteke(f::fa) :: {:ok, c::any} | :error
    # Egy nemüres f fa bal oldali szélső címkéje c (minden
    # felmenőjére is  igaz, hogy bal oldali gyermek)

   fa_balerteke(t1) === {:ok, 3}
   fa_balerteke(:level) === :error

-------------------------------------------------------------------------------
 8. Bináris fa jobb szélső címkéjének meghatározása

    @spec fa_jobberteke(f::fa) :: {:ok, c::any} | :error
    # Egy nemüres f fa jobb oldali szélső címkéje c (minden
    # felmenőjére is igaz, hogy jobb oldali gyermek)

   fa_jobberteke(t1) === {:ok, 7}
   fa_jobberteke(:level) === :error

-------------------------------------------------------------------------------
 9. Bináris fa rendezettsége

   Egy bináris fa rendezett, ha inorder bejárásakor a címkéi szigorúan monoton
   növekednek, azaz a csomópontjai kielégítik a keresőfa-tulajdonságot: minden
```

egyes csomópont címkéje nagyobb a bal oldali gyermekei címkéjénél és kisebb a jobb oldali gyermekei címkéjénél.

```
@spec rendezett_fa_2(f::fa) :: b::boolean
# b igaz, ha az f fa rendezett
```

a) Oldja meg a 7. és 8. feladatok szerinti fa_balerteke/1 és fa_jobberteke/1 függvényekkel;
b) oldja meg a 6. feladat szerinti cimkek/1 és az Enum.sort/1 függényekkel.

```
rendezett_fa(t1) === true
rendezett_fa(t2) === false
```

---

10. 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 :: [any]
@spec utak(f::fa) :: cimkezett_utak::[{c::any, cu::ut}]
# A cimkezett_utak lista az f fa minden csomópontjához egy {c,cu} párt
# társít, ahol c az adott csomópont címkéje, cu pedig az adott
# csomóponthoz vezető útvonal

utak(t1) === [{4,[]},{3,[4]},{6,[4]},{5,[4,6]},{7,[4,6]}]
utak(t2) === [{:a,[]},
              {:b,[:a]},
              {:v,[:a,:b]},
              {:c,[:a]},
              {:d,[:a,:c]},
              {:w,[:a,:c,:d]},
              {:x,[:a,:c,:d,:w]},
              {:f,[:a,:c,:d]},
              {:x,[:a,:c,:d,:f]},
              {:y,[:a,:c,:d,:f]}]
```

Javasolt segédfüggvény:

```
@spec utak(f::fa, eddigi::ut) :: cimkezett_utak::[{c::any, u::ut}]
# A cimkezett_utak lista az f fa minden csomópontjához egy {c,u} párt
# társít, ahol c az adott csomópont címkéje, u pedig az adott
# csomóponthoz vezető útvonal az eddigi eddigi útvonal elé fűzve
```

---

11. Címke összes előfordulása bináris fában útvonallal

```
@spec cutak(f::fa, c::any) :: utak::[{c::any, cu::ut}]
# utak azon csomópontok útvonalainak listája f-ben, amelyek címkéje c
```

a) Oldja meg listanézettel és a 10. feladat szerinti utak/1 felhasználásával;
b) oldja meg memóriatakarékosabban úgy, hogy csak a keresett útvonalakat tárolja az összes útvonal helyett.

```
cutak(t1, :x) === []
cutak(t2, :x) === [{:x,[:a,:c,:d,:w]},{:x,[:a,:c,:d,:f]}]
```

A b) pont szerinti megoldás nagyon hasonló lehet a 10. feladat megoldásához (vö. utak/2 segédfüggvény), de a fa gyökerének címkéjét csak egy bizonyos feltétel teljesülésekor tárolja el.

LUSTA FARKÚ LISTA

A feladatsorban a lazy_list adattípust a következő módon definiáljuk:

```
@type lazy_list :: nil | {any, (() -> lazy_list)}
# Ez a lista félig lusta: a fej mindig kiértékelődik, a farok lusta
```

A feladatsorban felhasználható generátorfüggvények:

```
  @spec seq(m::integer, n::integer) :: ll::lazy_list
  # Az m-től n-ig egyesével növekedő egész számok lusta farkú listája ll
  def seq(m, n) when m <= n, do: {m, fn() -> seq(m+1, n) end}
  def seq(_, _),             do: nil

  @spec infseq(n::integer) :: ll::lazy_list
  # Az n-nel kezdődő, egyesével növekedő egész számok lusta farkú listája ll
  def infseq(n), do: {n, fn() -> infseq(n+1) end}
```

---

12. Lusta farkú lista n-edik eleme

   a) `@spec nth(ll::lazy_list, n::integer) :: x::any`
      `# Az ll lusta farkú lista n-edik eleme x (számozás 0-tól)`

   b) `@spec nth_2(ll::lazy_list, n::integer) :: {:ok, x::any} | :error`
      `# Az ll lusta farkú lista n-edik eleme x (számozás 0-tól); a`
      `# visszatérési érték az :error atom, ha az ll lista üres, vagy`
      `# nincs több eleme, vagy ha n < 1`

```
    nth(infseq(0), 99) === 99
    try do nth(seq(0,5), 6) catch _,_ -> "nincs" end == "nincs"
    nth_2(infseq(0), 99) === {:ok, 99}
    nth_2(nil, 5) === :error
    nth_2(infseq(0), -1) === :error
    nth_2(seq(0,5), 6) === :error
```

---

13. Fibonacci-sorozat lusta farkú listával

   a) `@spec fibs(curr::integer, next::integer) :: ll::lazy_list`
      `# ll egy olyan általánosított Fibonacci-sorozat (lusta farkú lista),`
      `# amelynek első két tagja curr és next`

   b) `@spec fib(n::integer) :: f::integer`
      `# f az n-edik fibonacci-szám`

```
   nth(fibs(0,1), 99)   === 218922995834555169026
   nth_2(fibs(0,1), 99) === {:ok,218922995834555169026}
   fib(99) === 218922995834555169026
```

----- $LastChangedDate: 2020-11-11 12:43:16 +0100 (sze, 11 nov 2020) $ ------