```
defmodule Dp.Gy3 do

  @moduledoc """
  FP 3 gyakorlat

  @author "hanak@emt.bme.hu"
  @date   "$LastChangedDate: 2021-09-23 07:45:58 +0200 (cs, 23 szept 2021) $$"
  """
#-------------------------------------------------------------------------
#                           BINÁRIS FÁK
#-------------------------------------------------------------------------
#
@type fa      :: :level | {any, fa, fa}
@type egeszfa :: :level | {integer, egeszfa, egeszfa}
#
# 1.
@spec fa_noveltje(f0::egeszfa) :: f::egeszfa
# Az f fa minden címkéje eggyel nagyobb az f0 fa azonos helyen lévő címkéjénél
def fa_noveltje(:level),       do: :level
def fa_noveltje({c,bfa,jfa}), do: {c+1, fa_noveltje(bfa), fa_noveltje(jfa)}

# 2.
@spec fa_tukorkepe(f0::fa) :: f::fa
# f az f0 fa tükörképe
def fa_tukorkepe(:level),      do: :level
def fa_tukorkepe({c,bfa,jfa}), do: {c, fa_tukorkepe(jfa), fa_tukorkepe(bfa)}

# 3.
@spec inorder(f::fa) :: ls::[any]
# ls az f fa elemeinek a fa inorder bejárásával létrejövő listája
def inorder(:level),      do: []
def inorder({c,bfa,jfa}), do: inorder(bfa) ++ [c|inorder(jfa)]

@spec preorder(f::fa) :: ls::[any]
# ls az f fa elemeinek a fa preorder bejárásával létrejövő listája
def preorder(:level), do: []
def preorder({c,bfa,jfa}), do: [c|preorder(bfa) ++ preorder(jfa)]

@spec postorder(f::fa) :: ls::[any]
# ls az f fa elemeinek a fa postorder bejárásával létrejövő listája
def postorder(:level),      do: []
def postorder({c,bfa,jfa}), do: postorder(bfa) ++ (postorder(jfa) ++ [c])

# 4.
@spec tartalmaz(f::fa, c::any) :: b::boolean
# b igaz, ha c az f fa valamely címkéje
def tartalmaz(:level, _),      do: false
def tartalmaz({c,_,_}, c),     do: true
def tartalmaz({_,bfa,jfa}, c), do: tartalmaz(bfa, c) or tartalmaz(jfa, c)

# 5.
@spec elofordul(f::fa, c::any) :: n::integer
# A c címke az f fában n-szer fordul elő
def elofordul(:level, _),      do: 0
def elofordul({r,bfa,jfa}, c), do:
  # \ vagy + kell a sor végére, különben szintaktishiba!
  if r === c, do: 1, else: 0 \
   + elofordul(bfa, c) + elofordul(jfa, c)

# 6.
@spec cimkek(f::fa) :: ls::[any]
# ls az f címkéinek listája inorder sorrendben
def cimkek(fa), do: cimkek(fa, [])

@spec cimkek(f::fa, zs::[any]) :: ls::[any]
# ls az f címkéinek listája inorder sorrendben zs elé fűzve
def cimkek(:level, zs),      do: zs
def cimkek({r,bfa,jfa}, zs), do: cimkek(bfa, [r|cimkek(jfa, zs)])

# 7.
```

```
@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)
def fa_balerteke(:level),       do: :error
def fa_balerteke({c,:level,_}), do: {:ok, c}
def fa_balerteke({_,bfa,_}),    do: fa_balerteke(bfa)

# 8.
@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)
def fa_jobberteke(:level),       do: :error
def fa_jobberteke({c,_,:level}), do: {:ok, c}
def fa_jobberteke({_,_,jfa}),    do: fa_jobberteke(jfa)

# 9. a)
@spec rendezett_fa_2(f::fa) :: b::boolean
# b igaz, ha az f fa rendezett
def rendezett_fa_2(:level),     do: true
def rendezett_fa_2({c,bfa,jfa}) do
  case fa_jobberteke(bfa) do
    :error   -> true
    {:ok, j} -> j < c
  end
  and
  rendezett_fa_2(bfa)
  and
  case fa_balerteke(jfa) do
    :error   -> true
    {:ok, b} -> c < b
  end
  and
  rendezett_fa_2(jfa)
end

# 9. b)
@spec rendezett_fa(f::fa) :: b::boolean
# b igaz, ha az f fa rendezett
def rendezett_fa(f) do
  ls = cimkek(f)
  ls === Enum.sort(ls)
end

# 10.
@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
def utak(fa), do: utak(fa, [])

@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
def utak(:level, _),          do: []
def utak({c,bfa,jfa}, eddigi) do
  #  eddigi1 = eddigi ++ [c] # Költséges művelet: O(n2)!
  eddigi1 =  Enum.reverse([c | Enum.reverse(eddigi)]) # Kevésbé: O(2n)
  [{c, eddigi} | utak(bfa, eddigi1)] ++ utak(jfa, eddigi1)
end

# 11. a)
@spec cutak(f::fa, c::any) :: utak::[{c::any, cu::ut}]
# utak azon csomópontok útvonalainak listája f-ben, amelyek címkéje c
def cutak(fa, c), do: for {c0,_} = cu <- utak(fa), c0 === c, do: cu

# 11. b)
@spec cutak_2(f::fa, c::any) :: utak::[{c::any, cu::ut}]
# utak azon csomópontok útvonalainak listája f-ben, amelyek címkéje c
```

```
def cutak_2(fa, c), do: cutak_2(fa, c, [])

@spec cutak_2(f::fa, c::any, eddigi::ut) :: cimkezett_utak::[{c::any, u::ut}]
# A cimkezett-utak lista az f fa minden c címkéjű 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
def cutak_2(:level, _, _),          do: []
def cutak_2({r,bfa,jfa}, c, eddigi) do
  eddigi1 = eddigi ++ [r]
  cutak = cutak_2(bfa, c, eddigi1) ++ cutak_2(jfa, c, eddigi1)
  if r === c, do: [{c, eddigi} | cutak], else: cutak
end

#-------------------------------------------------------------------------
#                                LUSTA FARKÚ LISTA
#-------------------------------------------------------------------------

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

@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. a)
@spec nth(ll::lazy_list, n::integer) :: x::any
# Az ll lusta farkú lista n-edik eleme x (számozás 0-tól)
def nth({h, _t}, 0), do: h
def nth({_h, t}, n), do: nth(t.(), n-1)

# 12. 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
def nth_2({h, _t}, 0),            do: {:ok, h}
def nth_2({_h, t}, n) when n > 0, do: nth_2(t.(), n-1)
def nth_2(_, _),                  do: :error

# 13. 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
def fibs(curr, next), do: {curr, fn() -> fibs(next, curr+next) end}

# 13. b)
@spec fib(n::integer) :: f::integer
# f az n-edik fibonacci-szám
def fib(n), do: nth(fibs(0, 1), n)
#
#-------------------------------------------------------------------------
#
@spec fm_list_in([any]) :: fa
# Listából fát épít inorder sorrendben
def fm_list_in([]), do: :level
def fm_list_in(ls) do
  {xs, [y|ys]} = Enum.split(ls, div(length(ls), 2))
  {y, fm_list_in(xs), fm_list_in(ys)}
end

def test(:def) do
  t0 =
  {1,
   {2,
    {4,:level,:level},
```

```
    {5,:level,:level}
   },
   {3,:level,:level}
  }
  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}
     }
    }
   }
  }
  t3 =
  {4,
   {3,
    {2,:level,:level},
    {1,:level,:level}
   },
   {6,
    {5,:level,:level},
    {7,:level,:level}
   }
  }
  {t0,t1,t2,t3}
end

def test(:fa1) do
  {t0,t1,t2,_t3} = test(:def)
  [fa_noveltje(t1) ===
     {5,{4,:level,:level},{7,{6,:level,:level},{8,:level,:level}}},
   fa_tukorkepe(t1) ===
     {4,{6,{7,:level,:level},{5,:level,:level}},{3,:level,:level}},
   inorder(t0)   === [4,2,5,1,3],
   preorder(t0)  === [1,2,4,5,3],
   postorder(t0) === [4,5,2,3,1],
   inorder(t1)   === [3,4,5,6,7],
   preorder(t1)  === [4,3,6,5,7],
   postorder(t1) === [3,5,7,6,4],
   tartalmaz(t1, :x) === false,
   tartalmaz(t2, :x) === true,
   elofordul(t1, :x) === 0,
   elofordul(t2, :x) === 2,
   cimkek(t1) === [3,4,5,6,7]
  ]
end

def test(:fa2) do
    {_t0,t1,t2,_t3} = test(:def)
    [fa_balerteke(t1)      === {:ok, 3},
     fa_balerteke(:level)  === :error,
     fa_balerteke(t2)      === {:ok, :v},
     fa_jobberteke(t1)     === {:ok, 7},
     fa_jobberteke(t2)     === {:ok, :y},
     fa_jobberteke(:level) === :error,
     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]}],
     cutak(t1,:x)   === [],
     cutak(t2,:x)   === [{:x,[:a,:c,:d,:w]},{:x,[:a,:c,:d,:f]}],
     cutak_2(t1,:x) === cutak(t1,:x),
     cutak_2(t2,:x) === cutak(t2,:x)
    ]
end

def test(:fa3) do
    {_t0,t1,t2,t3} = test(:def)
    [rendezett_fa(t1) === true,
     rendezett_fa(t2) === false,
     rendezett_fa(t3) === false,
     rendezett_fa_2(t1) === true,
     rendezett_fa_2(t2) === false,
     rendezett_fa_2(t3) === false,
     rendezett_fa(fm_list_in([1,2,3,4,5,6,7,8])) === true,
     rendezett_fa(fm_list_in([-3.4,-1.2,2.0,3,4.5,5,6.8,7.9,8])) === true,
     rendezett_fa(fm_list_in([:a,:b,:c,:d,:e,:f,:g,:h])) === true,
     rendezett_fa(fm_list_in('abcdefgh')) === true,
     rendezett_fa(fm_list_in([])) === true,
     rendezett_fa(fm_list_in([1,1.0,:atom,&nth/2,{3},{1,2},[5,:x],[5.01]])) === true,
     rendezett_fa_2(fm_list_in([1,2,3,4,5,6,7,8])) === true,
     rendezett_fa_2(fm_list_in([-3.4,-1.2,2.0,3,4.5,5,6.8,7.9,8])) === true,
     rendezett_fa_2(fm_list_in([:a,:b,:c,:d,:e,:f,:g,:h])) === true,
     rendezett_fa_2(fm_list_in('abcdefgh')) === true,
     rendezett_fa_2(fm_list_in([])) === true,
     rendezett_fa_2(fm_list_in([1,1.01,:atom,&nth/2,{3},{1,2},[5,:x],[5.01]])) === true
    ]
end

def test(:lusta) do
  [ nth(infseq(0),99) === 99,
    try do nth(seq(0,5), 6) catch _,_ -> "Nemlétező" end == "Nemlétező",
    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,
    nth(fibs(0,1), 99) === 218922995834555169026,
    nth_2(fibs(0,1), 99) === {:ok,218922995834555169026},
    nth_2(fibs(0,1), -1) === :error,
    fib(99) === 218922995834555169026
  ]
end

end

IO.inspect Dp.Gy3.test(:fa1)
IO.inspect Dp.Gy3.test(:fa2)
IO.inspect Dp.Gy3.test(:fa3)
IO.inspect Dp.Gy3.test(:lusta)
```