Tartalom

DP25a, mintazéhá, 2025. október

Személyi adatok

Az Elixir-cellában lévő kódba írja be a kért adatokat: név, NEPTUN-kód és hely (pl 307b16).

```
defmodule Personal do
 @nev ""
 @neptun ""
 @hely ""

 def check() do
  unless @nev != "" and @neptun != "" and @hely != ""do
   "Írja be a kért adatokat!"
  else
   "Köszönjük az adatok beírását!"
  end
 end
end
Personal.check()
```

```
defmodule Personal do
 @nev ""
 @neptun ""
 @hely ""

 def check() do
  unless @nev != "" and @neptun != "" and @hely != "" do
   "Írja be a kért adatokat!"
  else
   "Köszönjük az adatok beírását!"
  end
 end
end
Personal.check()
```

A virtuális gépről és környezetről

A zárthelyit Oracle Virtualboxban futó Ubuntu 24.04 operációs rendszer alatt íratjuk. A felhasználó neve dp, jelszava dpfpzarthelyi. A virtuális gép indulásakor a belépés automatikus, a Livebook is elindul, a jelszó azonos a dp felhasználó jelszavával.

Ha a Livebook nem futna, egy virtuális terminálban

- indítsa el a /home/ubuntu/run_livebook.sh szkriptet, vagy
- adja ki a livebook server lb parancsot – az lb mappanév megadása fontos a böngészőablak automatikus megnyitásához.

A Livebook szerver indítása elindítja a Firefox böngészőt is. A Livebookon kívül két további fül is megnyílik. Az egyik az <a href="https://dps.iit.bme.hu/ets" _target=blank>Elektronikus TanárSegéd, ETS , a másik egy kivonat az <a href="https://dps.iit.bme.hu/exdoc" _target=blank>Elixir dokumentációból. Más szervert nem lehet elérni a virtuális gépről.

Bármilyen más online felület használata TILOS, bármilyen eszközről!

A beadásról

A zárthelyi megoldását az ETS-sel kell beadni. Csak olyan .exs fájlt adjon be, ami a Livebookban futtatható, akkor is, ha az eredmény hibás.

1. A megoldás elmentéséhez kattintson a lap tetején a bal felső sarokban található három függőleges pontra, majd az *Export* menüpontra. Válassza ki az *IEX session* fület, majd kattintson a *Download source* ikonra. Az elmentett fájl (valószínűleg) a böngésző által használt Downloads/Letöltések mappába kerül.
2. Lépjen be az ETS-be. A zárthelyi megoldását a *Feladatbeadás* menüpont *Elixir …zéhá* sorában kell feltöltenie. A feltöltendő fájl neve bármi lehet, az ETS át fogja nevezni zh1.exs-re.
3. Ha a beadott .exs fájlt az ETS nem tudja lefordítani, javítsa ki a hibát, és adja be újra. A legutolsó beadást tekintjük érvényesnek.
4. A beadott megoldást akkor is értékeljük – pontlevonások mellett – ha nem sikerült hibátlanul lefordúló programot feltöltenie.

Fontos! Nem a *.livemd, hanem a *.exs fájlt kell felölteni!

1. feladat: Egyklózos, nemrekurzív és magasabb rendű függvények

írjon egy-egy egyklózos, nemrekurzív függvényt az alábbi feladatok (f1, f2, f3) megoldására. Segédfüggvényt nem definiálhat.

Javasoljuk a megfelelő helyeken a for-komprehenzió, valamint az Enum.zip/2 és az Enum.split_while/2 használatát.

```
defmodule T1 do

 def f(i, xs) do

  # @spec f1(xs :: [any()]) :: rs :: [any()]
  # Az xs lista hárommal *nem* osztható indexű elemeiből álló lista zs, az
  # elemek eredeti sorrendjében. Indexelés 0-tól. Az xs lista üres is lehet.       (3 pont)
  f1 = fn(xs) -> _ = xs; nil end

  # @spec f2(coll :: [integer()] | Range.t()) :: s :: integer()
  # A coll számlista vagy tartomány páros értékű elemeinek összege s              (3 pont)
  f2 = fn(coll) -> _ = coll; nil end

  # @spec f3(xs::[integer()]) :: n::integer()
  # Az xs lista monoton növekvő elemekből álló, lehető leghosszabb prefixuma n
  # hosszú. Azaz xs első n eleme monoton növekvő sorozatot alkot, és az xs vagy
  # pontosan n elemből áll, vagy az (n+1)-edik eleme kisebb az előtte
  # állónál. Feltételezheti, hogy xs nem üres.                                    (4 pont)
  f3 = fn(xs) -> _ = xs; nil end

   case i do
    1 -> f1.(xs)
    2 -> f2.(xs)
    3 -> f3.(xs)
   end

 end
end

(T1.f(1, [1,-4,5,6,"5",-3,12,11,7,4]) === [-4,5,"5",-3,11,7]) |> IO.inspect()
(T1.f(1, [:a,4])                       === [4])                |> IO.inspect()
(T1.f(1, [:b])                         === [])                 |> IO.inspect()
(T1.f(1, [0,:c,-1])                    === [:c,-1])            |> IO.inspect()
(T1.f(1, [])                           === [])                 |> IO.inspect()

(T1.f(2, 1..10 // 1) === 30)                                   |> IO.inspect()
(T1.f(2, 10..1 // -3) === 14)                                  |> IO.inspect()
(T1.f(2, [-7, -4, -2, -1, 0, 4, 9, 12, 14, 17, 26]) === 50) |> IO.inspect()

(T1.f(3, [6]) === 1)                          |> IO.inspect()
(T1.f(3, [6, 7, 8, 6, 7, 8, 6, 7, 9]) === 3) |> IO.inspect()
(T1.f(3, [2, 2, 2, 3, 3, 3]) === 6)           |> IO.inspect()
```

```
(T1.f(3, [-1, -2, -2]) === 1)            |> IO.inspect()
(T1.f(3, [3, 2, 1, 0]) === 1)            |> IO.inspect()
(T1.f(3, [1, 2, 3, 4, 5, 6, 7]) === 7)     |> IO.inspect()


(T1.f(1, [1,-4,5,6,"5",-3,12,11,7,4]) === [-4,5,"5",-3,11,7]) |> IO.inspect()
(T1.f(1, [:a,4])                 === [4])          |> IO.inspect()
(T1.f(1, [:b])                   === [])           |> IO.inspect()
(T1.f(1, [0,:c,-1])               === [:c,-1])       |> IO.inspect()
(T1.f(1, [])                   === [])           |> IO.inspect()

(T1.f(2, 1..10 // 1) === 30)                        |> IO.inspect()
(T1.f(2, 10..1 // -3) === 14)                       |> IO.inspect()
(T1.f(2, [-7, -4, -2, -1, 0, 4, 9, 12, 14, 17, 26]) === 50) |> IO.inspect()

(T1.f(3, [6]) === 1)                  |> IO.inspect()
(T1.f(3, [6, 7, 8, 6, 7, 8, 6, 7, 9]) === 3) |> IO.inspect()
(T1.f(3, [2, 2, 2, 3, 3, 3]) === 6)       |> IO.inspect()
(T1.f(3, [-1, -2, -2]) === 1)            |> IO.inspect()
(T1.f(3, [3, 2, 1, 0]) === 1)            |> IO.inspect()
(T1.f(3, [1, 2, 3, 4, 5, 6, 7]) === 7)     |> IO.inspect()
```

2. feladat: Zsákok metszete

A zsák (angolul bag vagy multiset) olyan adatstruktúra, amelyben az elemek sorrendje – a halmazhoz hasonlóan – érdektelen, ám – a halmazzal ellentétben – ugyanaz az elem többször is előfordulhat benne. Egy elem előfordulásainak számát multiplicitásnak nevezzük.

Elixirben egy zsákot pl. map-pel ábrázolhatunk, ahol az elem a kulcs, az érték pedig a multiplicitása. Az kulcs tetszőleges típusú érték lehet, a multiplicitás pozitív egész szám.

Írjon olyan függvényt **metszet** néven, amely két zsák metszetét adja eredményül. A zsákok üresek is lehetnek.

Nem használhatja a Map.intersect/3 függvényt. Tipp: a for komprehenzióval képzett keresztszorzat (Descartes-szorzat) hasznos lehet...

```
defmodule T2 do
 @type elem() :: %{any(), integer()}
 @spec metszet(am::elem(), bm::elem()) :: cm::elem()
 # Az am és bm zsákok metszete a cm zsák, azaz a cm kizárólag az am és bm közös elemeit
 # tartalmazza, mégpedig úgy, hogy cm minden elemének multiplicitása az adott elem
 # am-beli és bm-beli multiplicitása közül a kisebbik.
 # Az am és bm zsákok üresek is lehetnek.                    (11 pont)

 def metszet(am, bm), do: (_ = am; _ = bm; %{})

end
(T2.metszet(%{}, %{}) === %{})                        |> IO.inspect()
(T2.metszet(%{}, %{111=>10}) === %{})                    |> IO.inspect()
(T2.metszet(%{11=>10,33=>30}, %{22=>20,44=>40}) === %{})          |> IO.inspect()
(T2.metszet(%{11=>10},22=>2},33=>30}, %{11=>1,22=>20}) === %{11=>1,22=>2}) |> IO.inspect()


(T2.metszet(%{}, %{}) === %{})                        |> IO.inspect()
(T2.metszet(%{}, %{111=>10}) === %{})                    |> IO.inspect()
(T2.metszet(%{11=>10,33=>30}, %{22=>20,44=>40}) === %{})          |> IO.inspect()
(T2.metszet(%{11=>10,22=>2,33=>30}, %{11=>1,22=>20}) === %{11=>1,22=>2}) |> IO.inspect()
```

3. feladat: Listakezelés rekurzív függvénnyel: lejtők

Egy egészlistában lejtőnek nevezzük a legalább két elemű, egyik irányban sem kiterjeszthető, folytonos, szigorúan monoton csökkenő sorozatokat. Írjon olyan rekurzív függvényt **lejtok** néven az alább definiált T3.lejto/2 segédfüggvény felhasználásával, amely a paraméterként átadott listából kigyűjti a lejtőket alkotó részlistákat, és ezek listáját adja eredményül, az eredeti sorrend megtartásával. Saját segédfüggvényt definiálhat.

A legalább kételemű listákat mintaillesztéssel ismerje fel, ehhez ne használjon when kifejezést őrrel, se a lista hosszát meghatározó bármilyen függvényt, pl. a length/1-et. Nem használhatja az Enum.chunk_by/2, Enum.chunk_every/2, Enum.chunk_every/4 és Enum.chunk_while/4 függvényeket sem.

```
defmodule T3 do

 @spec lejto(xs::[integer()], zs::[z::integer()]) :: {ls::[integer()], rs::[integer()]}
 # Az [z|xs] egészlista balról az első, lehető leghosszabb, folytonos,
 # szigorúan monoton csökkenő sorozata ls, maradéka rs
 def lejto([x|xs], [z|_]=zs) when z > x, do: lejto(xs, [x|zs])
 def lejto(xs, zs), do: {Enum.reverse(zs), xs}

 @spec lejtok(xs::[integer()]) :: rss::[[integer()]]
 # Az rss az xs-ben előforduló lejtők listája
 # Lejtőnek nevezzük a legalább két elemű, egyik irányban sem kiterjeszthető,
 # folytonos, szigorúan monoton csökkenő sorozatot.                    (10 pont)

 def lejtok(xs) do _ = xs; nil end

end

(T3.lejtok([0, 0, 0, 0]) === [])                |> IO.inspect()
(T3.lejtok([-1, -2, -3, -4]) === [[-1, -2, -3, -4]])  |> IO.inspect()
(T3.lejtok([]) === [])                          |> IO.inspect()
(T3.lejtok([5]) === [])                         |> IO.inspect()
(T3.lejtok([5, 4]) === [[5, 4]])                |> IO.inspect()
(T3.lejtok([4, 0, 6, 0, 9, 18]) === [[4, 0], [6, 0]]) |> IO.inspect()


(T3.lejtok([0, 0, 0, 0]) === [])                |> IO.inspect()
(T3.lejtok([-1, -2, -3, -4]) === [[-1, -2, -3, -4]])  |> IO.inspect()
(T3.lejtok([]) === [])                          |> IO.inspect()
(T3.lejtok([5]) === [])                         |> IO.inspect()
(T3.lejtok([5, 4]) === [[5, 4]])                |> IO.inspect()
(T3.lejtok([4, 0, 6, 0, 9, 18]) === [[4, 0], [6, 0]]) |> IO.inspect()
```

4. feladat: Bináris fában lévő számok mediánja

Egy bináris fa típusát így definiáljuk:

```
@type bintree() :: :l | {bintree(), v::any(), bintree()}
```

Írjon függvényt median néven egy bináris fában lévő számok medián (helyzeti közép-)értékének visszaadására! Ha a fában lévő számok száma páros, a két középső érték átlaga legyen az eredmény. Ha a fában nincs szám, a függvény visszatérési értéke nil legyen. Segédfüggvényt definiálhat.

```
defmodule T4 do

 @type bintree() :: :l | {bintree(), v :: any(), bintree()}
 @spec median(t :: bintree()) :: med :: float() | nil
 # a t fában lévő számok átlaga med                          (12 pont)
 def median(t) do _ = t; nil end
 # Megoldás
 #  def median(t) do
 #    case to_list(t, []) do
  #     [_ | _] = xs -> xs |> Enum.sort() |> calc_med()
  #     [] -> nil
 #    end
 #  end

 #  def calc_med(xs) do
 #    len = length(xs)
 #    idx = div(len,2)
 #    case rem(len,2) do
  #     1 -> Enum.at(xs, idx)
```

```
  #     0 -> (Enum.at(xs, idx-1) + Enum.at(xs, idx)) / 2
  #   end
  # end

  # def to_list(:l, vs), do: vs
  # def to_list({lt, v, rt}, vs) do
  #   to_list(lt, to_list(rt, if(is_integer(v), do: [v | vs], else: vs)))
  # end

end

t0a = {:l, 8, :l}
t0b = {:l, :a, :l}
t1a = {{:l, 5, :l}, 6, {:l, 4, :l}}
t1b = {{:l, 5, :l}, :a, {:l, 4, :l}}
t1c = {{:l, :b, :l}, :a, {:l, :c, :l}}
t2a = {{{:l, :a, :l}, 8, {:l, 5, :l}}, 7, {:l, 6, :l}}
t2b = {{{:l, 9, :l}, :a, :l}, 3, {:l, 6, :l}}
t3a = {{{{{{{:l, 11, :l}, 7, {:l, 12, :l}}, 13, :l}, 9, :l}, 3, :l}, 5, :l}, 4, :l}
t3b = {{{{{{{:l, :x, :l}, 7, {:l, 12, :l}}, 13, :l}, 1, :l}, 5, :l}, 4, :l}, :a, :l}

(T4.median(t0a) == 8.0) |> IO.inspect()
(T4.median(t0b) == nil) |> IO.inspect()
(T4.median(t1a) == 5.0) |> IO.inspect()
(T4.median(t1b) == 4.5) |> IO.inspect()
(T4.median(t1c) == nil) |> IO.inspect()
(T4.median(t2a) == 6.5) |> IO.inspect()
(T4.median(t2b) == 6.0) |> IO.inspect()
(T4.median(t3a) == 8.0) |> IO.inspect()
(T4.median(t3b) == 6.0) |> IO.inspect()


t0a = {:l, 8, :l}
t0b = {:l, :a, :l}
t1a = {{:l, 5, :l}, 6, {:l, 4, :l}}
t1b = {{:l, 5, :l}, :a, {:l, 4, :l}}
t1c = {{:l, :b, :l}, :a, {:l, :c, :l}}
t2a = {{{:l, :a, :l}, 8, {:l, 5, :l}}, 7, {:l, 6, :l}}
t2b = {{{:l, 9, :l}, :a, :l}, 3, {:l, 6, :l}}
t3a = {{{{{{{:l, 11, :l}, 7, {:l, 12, :l}}, 13, :l}, 9, :l}, 3, :l}, 5, :l}, 4, :l}
t3b = {{{{{{{:l, :x, :l}, 7, {:l, 12, :l}}, 13, :l}, 1, :l}, 5, :l}, 4, :l}, :a, :l}

(T4.median(t0a) == 8.0) |> IO.inspect()
(T4.median(t0b) == nil) |> IO.inspect()
(T4.median(t1a) == 5.0) |> IO.inspect()
(T4.median(t1b) == 4.5) |> IO.inspect()
(T4.median(t1c) == nil) |> IO.inspect()
(T4.median(t2a) == 6.5) |> IO.inspect()
(T4.median(t2b) == 6.0) |> IO.inspect()
(T4.median(t3a) == 8.0) |> IO.inspect()
(T4.median(t3b) == 6.0) |> IO.inspect()
```