Tartalom

Csúszóablakos technika, kihagy-bevesz rekurzió

```
Mix.install([
{:benchee, "~> 1.3"}
])
```

Csúszóablakos problémamegoldási technika

Számlista elejétől kezdődő folytonos részlistái

```
defmodule Reszlistak0 do
 def reszlistak([x|xs]), do: reszlistak(xs, [x], [[x]])
 def reszlistak([]), do: []

 def reszlistak([y|ys], ss, zss) do
  ss_uj = [y|ss]
  reszlistak(ys, ss_uj, [Enum.reverse(ss_uj) | zss])
 end
 def reszlistak([], _ss, zss), do: zss
end
```

```
{:module, Reszlistak0, <<70, 79, 82, 49, 0, 0, 9, ...>>, {:reszlistak, 3}}
```

```
Reszlistak0.reszlistak([1,2,3,4,5]) |> Enum.reverse()
```

```
[[1], [1, 2], [1, 2, 3], [1, 2, 3, 4], [1, 2, 3, 4, 5]]
```

Számlista összes folytonos részlistája

```
defmodule Reszlistak1 do
 def reszlistak([_x|_xs]=xxs), do: reszlistak(xxs, [])

 def reszlistak([_x|xs]=xxs, zss) do
  reszlistak(xs, Reszlistak0.reszlistak(xxs) ++ zss)
 end

 def reszlistak([], zss), do: zss
end
```

```
{:module, Reszlistak1, <<70, 79, 82, 49, 0, 0, 9, ...>>, {:reszlistak, 2}}
```

```
Reszlistak1.reszlistak([1,2,3,4,5]) |> Enum.reverse()
```

```
[
 [1],
 [1, 2],
 [1, 2, 3],
 [1, 2, 3, 4],
 [1, 2, 3, 4, 5],
 [2],
 [2, 3],
```

```
 [2, 3, 4],
 [2, 3, 4, 5],
 [3],
 [3, 4],
 [3, 4, 5],
 [4],
 [4, 5],
 [5]
]
```

Számlista összes folytonos részlistája és ezek összege

```
defmodule Reszlistak2 do
 def reszlistak(xs), do: reszlistak(Reszlistak1.reszlistak(xs), [])

 def reszlistak([xs|xss], zss), do: reszlistak(xss, [{Enum.sum(xs), xs} | zss])
 def reszlistak([], zss), do: zss
end
```

```
{:module, Reszlistak2, <<70, 79, 82, 49, 0, 0, 8, ...>>, {:reszlistak, 2}}
```

```
Reszlistak2.reszlistak([1,2,3,4,5]) |> Enum.reverse()
```

```
[
 {5, [5]},
 {9, [4, 5]},
 {4, [4]},
 {12, [3, 4, 5]},
 {7, [3, 4]},
 {3, [3]},
 {14, [2, 3, 4, 5]},
 {9, [2, 3, 4]},
 {5, [2, 3]},
 {2, [2]},
 {15, [1, 2, 3, 4, 5]},
 {10, [1, 2, 3, 4]},
 {6, [1, 2, 3]},
 {3, [1, 2]},
 {1, [1]}
]
```

Számlista max. összegű folytonos részlistái

```
defmodule Reszlistak3 do
 def reszlistak(xs), do: reszlistak(Reszlistak1.reszlistak(xs), 0, [])

 def reszlistak([xs|xss], max, zss) do
  sum = Enum.sum(xs)
  cond do
   sum > max -> reszlistak(xss, sum, [xs])
   sum == max -> reszlistak(xss, max, [xs | zss])
   true -> reszlistak(xss, max, zss)
  end
 end
 def reszlistak([], max, zss), do: {max, zss}
end
```

```
{:module, Reszlistak3, <<70, 79, 82, 49, 0, 0, 10, ...>>, {:reszlistak, 3}}
```

```
Reszlistak3.reszlistak([1,2,3,4,5]) |> IO.inspect()
Reszlistak3.reszlistak([1,2,3,4,-10,4,3,2,1]) |> IO.inspect()
```

```
{15, [[1, 2, 3, 4, 5]]}
{10, [[1, 2, 3, 4], [1, 2, 3, 4, -10, 4, 3, 2, 1], [4, 3, 2, 1]]}
```

```
{10, [[1, 2, 3, 4], [1, 2, 3, 4, -10, 4, 3, 2, 1], [4, 3, 2, 1]]}
```

Számlista elejétől kezdődő folytonos részlistái és összegük

```
defmodule Reszlistak0ossz do
 def reszlistak([x|xs]), do: reszlistak(xs, [x], [{x, [x]}])
 def reszlistak([]), do: []

 def reszlistak([y|ys], ss, zss) do
  ss_uj = [y|ss]
  reszlistak(ys, ss_uj, [{Enum.sum(ss_uj), Enum.reverse(ss_uj)} | zss])
 end
 def reszlistak([], _ss, zss), do: zss
end
```

```
{:module, Reszlistak0ossz, <<70, 79, 82, 49, 0, 0, 9, ...>>, {:reszlistak, 3}}
```

```
Reszlistak0ossz.reszlistak([1,2,3,4,5]) |> Enum.reverse() |> IO.inspect()
Reszlistak0ossz.reszlistak([1,2,3,-3,-2,5]) |> Enum.reverse() |> IO.inspect()
```

```
[
 {1, [1]},
 {3, [1, 2]},
 {6, [1, 2, 3]},
 {10, [1, 2, 3, 4]},
 {15, [1, 2, 3, 4, 5]}
]
[
 {1, [1]},
 {3, [1, 2]},
 {6, [1, 2, 3]},
 {3, [1, 2, 3, -3]},
 {1, [1, 2, 3, -3, -2]},
 {6, [1, 2, 3, -3, -2, 5]}
]
```

```
[
 {1, [1]},
 {3, [1, 2]},
 {6, [1, 2, 3]},
 {3, [1, 2, 3, -3]},
 {1, [1, 2, 3, -3, -2]},
 {6, [1, 2, 3, -3, -2, 5]}
]
```

Számlista elejétől kezdődő, max. összegű folytonos részlistái

```
defmodule Reszlistak0max do
 def reszlistak([x|xs]), do: reszlistak(xs, [x], x, [[x]])
 def reszlistak([]), do: []

 def reszlistak([y|ys], ss, max, zss) do
  ss_uj = [y|ss]
  ss_uj_rev = Enum.reverse(ss_uj) # |> IO.inspect(label: "ss_uj_rev")
  sum = Enum.sum(ss_uj_rev) # |> IO.inspect(label: "sum")
  cond do
   sum > max -> reszlistak(ys, ss_uj, sum, [ss_uj_rev])
   sum == max -> reszlistak(ys, ss_uj, max, [ss_uj_rev | zss])
   true -> reszlistak(ys, ss_uj, max, zss)
  end
 end
 def reszlistak([], _ss, max, zss), do: {max, Enum.reverse(zss)}
end
```

```
{:module, Reszlistak0max, <<70, 79, 82, 49, 0, 0, 11, ...>>, {:reszlistak, 4}}
```

```
Reszlistak0max.reszlistak([1,2,3,-3,-2,5]) |> IO.inspect()
Reszlistak0max.reszlistak([6,-6,1,2,3,-3,-2,5]) |> IO.inspect()
```

```
{6, [[1, 2, 3], [1, 2, 3, -3, -2, 5]]}
{6, [[6], [6, -6, 1, 2, 3], [6, -6, 1, 2, 3, -3, -2, 5]]}
```

```
{6, [[6], [6, -6, 1, 2, 3], [6, -6, 1, 2, 3, -3, -2, 5]]}
```

```
defmodule Reszlistak0maxTake do
 def reszlistak(xs), do: reszlistak(xs, length(xs)-1, Enum.sum(xs), [xs])

 def reszlistak(_xs, 0, max, zss), do: {max, zss}
 def reszlistak(xs, len, max, zss) do
  ss = Enum.take(xs, len)
  sum = Enum.sum(ss)
  cond do
   sum > max -> reszlistak(xs, len-1, sum, [ss])
   sum == max -> reszlistak(xs, len-1, max, [ss | zss])
   true -> reszlistak(xs, len-1, max, zss)
  end
 end
end
```

```
{:module, Reszlistak0maxTake, <<70, 79, 82, 49, 0, 0, 11, ...>>, {:reszlistak, 4}}
```

```
Reszlistak0maxTake.reszlistak([1,2,3,-3,-2,5]) |> IO.inspect()
Reszlistak0maxTake.reszlistak([6,-6,1,2,3,-3,-2,5]) |> IO.inspect()
```

```
{6, [[1, 2, 3], [1, 2, 3, -3, -2, 5]]}
{6, [[6], [6, -6, 1, 2, 3], [6, -6, 1, 2, 3, -3, -2, 5]]}
```

```
{6, [[6], [6, -6, 1, 2, 3], [6, -6, 1, 2, 3, -3, -2, 5]]}
```

Számlista max. összegű folytonos részlistái

```
defmodule Reszlistak3max do
 # Reszlistak0max.reszlistak/1 segédfüggvény beégetve
 def reszlistak([_x|xs]=xxs), do: reszlistak(xs, Reszlistak0max.reszlistak(xxs))
 def reszlistak([]), do: {}

 def reszlistak([_y|ys]=yys, {maxsum, zss}) do
  {max, mss} = Reszlistak0max.reszlistak(yys)
  cond do
   max > maxsum -> reszlistak(ys, {max, mss})
   max == maxsum -> reszlistak(ys, {maxsum, zss ++ mss}) # hatékonyság vs. sorrend!
   true -> reszlistak(ys, {maxsum, zss})
  end
 end
 def reszlistak([], maxlists), do: maxlists
end
```

```
{:module, Reszlistak3max, <<70, 79, 82, 49, 0, 0, 10, ...>>, {:reszlistak, 2}}
```

```
Reszlistak3max.reszlistak([1,2,3,4,5]) |> IO.inspect()
Reszlistak3max.reszlistak([1,2,3,4,-10,4,3,2,1]) |> IO.inspect()
```

```
{15, [[1, 2, 3, 4, 5]]}
{10, [[1, 2, 3, 4], [1, 2, 3, 4, -10, 4, 3, 2, 1], [4, 3, 2, 1]]}
```

```
{10, [[1, 2, 3, 4], [1, 2, 3, 4, -10, 4, 3, 2, 1], [4, 3, 2, 1]]}
```

```
defmodule Reszlistak3mxrls do
 # Számlista elejétől kezdődő, folytonos, max. összegű részlistákat előállító
 # segédfüggvény mxrls paraméterként átadva
 def reszlistak(mxrls, [_x|xs]=xxs), do: reszlistak(mxrls, xs, mxrls.(xxs))
 def reszlistak(_mxrls, []), do: {}

 def reszlistak(mxrls, [_y|ys]=yys, {maxsum, zss}) do
  {max, mss} = mxrls.(yys)
  cond do
   max > maxsum -> reszlistak(mxrls, ys, {max, mss})
   max == maxsum -> reszlistak(mxrls, ys, {maxsum, zss ++ mss}) # hatékonyság vs. sorrend!
```

```
    true -> reszlistak(mxrls, ys, {maxsum, zss})
   end
  end
  def reszlistak(_mxrls, [], maxlists), do: maxlists
end
```

```
{:module, Reszlistak3mxrls, <<70, 79, 82, 49, 0, 0, 11, ...>>, {:reszlistak, 3}}
```

```
(&Reszlistak0max.reszlistak/1) |> Reszlistak3mxrls.reszlistak([1,2,3,4,5]) |> IO.inspect()
(&Reszlistak0max.reszlistak/1) |> Reszlistak3mxrls.reszlistak([1,2,3,4,-10,4,3,2,1]) |> IO.inspect()
(&Reszlistak0maxTake.reszlistak/1) |> Reszlistak3mxrls.reszlistak([1,2,3,4,5]) |> IO.inspect()
(&Reszlistak0maxTake.reszlistak/1) |> Reszlistak3mxrls.reszlistak([1,2,3,4,-10,4,3,2,1]) |> IO.inspect()
```

```
{15, [[1, 2, 3, 4, 5]]}
{10, [[1, 2, 3, 4], [1, 2, 3, 4, -10, 4, 3, 2, 1], [4, 3, 2, 1]]}
{15, [[1, 2, 3, 4, 5]]}
{10, [[1, 2, 3, 4], [1, 2, 3, 4, -10, 4, 3, 2, 1], [4, 3, 2, 1]]}
```

```
{10, [[1, 2, 3, 4], [1, 2, 3, 4, -10, 4, 3, 2, 1], [4, 3, 2, 1]]}
```

```
randomlist =
 (for n <- 1..10, do: (if rem(n,3)==0, do: -1, else: 1) * Enum.random(1..5))
 |> IO.inspect()
(&Reszlistak0max.reszlistak/1) |> Reszlistak3mxrls.reszlistak(randomlist)
```

```
[3, 3, -1, 2, 4, -3, 4, 3, -5, 3]
```

```
{15, [[3, 3, -1, 2, 4, -3, 4, 3]]}
```

Benchmarking: maximális összegű folytonos részlisták futási ideje

```
# xs = 1..1000 |> Enum.to_list
xs = for n <- 1..1000, do: (if rem(n,3)==0, do: -1, else: 1) * Enum.random(1..5)
Benchee.run(
 %{
  "utolag keres" =>
   fn -> Reszlistak3.reszlistak(xs) end,
  "menet kozben, sajat fv" =>
   fn -> Reszlistak3mxrls.reszlistak(&Reszlistak0max.reszlistak/1, xs) end,
  "menet kozben, Enum.take" =>
   fn -> Reszlistak3mxrls.reszlistak(&Reszlistak0maxTake.reszlistak/1, xs) end
 }# , profile_after: true
)
:ok
```

```
Warning: the benchmark menet kozben, Enum.take is using an evaluated function.
  Evaluated functions perform slower than compiled functions.
  You can move the Benchee caller to a function in a module and invoke `Mod.fun()` instead.
  Alternatively, you can move the benchmark into a benchmark.exs file and run mix run benchmark.exs

Warning: the benchmark menet kozben, sajat fv is using an evaluated function.
  Evaluated functions perform slower than compiled functions.
  You can move the Benchee caller to a function in a module and invoke `Mod.fun()` instead.
  Alternatively, you can move the benchmark into a benchmark.exs file and run mix run benchmark.exs

Warning: the benchmark utolag keres is using an evaluated function.
  Evaluated functions perform slower than compiled functions.
  You can move the Benchee caller to a function in a module and invoke `Mod.fun()` instead.
  Alternatively, you can move the benchmark into a benchmark.exs file and run mix run benchmark.exs

Operating System: Linux
CPU Information: Intel(R) Core(TM) i5-8365U CPU @ 1.60GHz
Number of Available Cores: 8
Available memory: 38.81 GB
Elixir 1.18.4
Erlang 28.0.1
JIT enabled: true
```

```
Benchmark suite executing with the following configuration:
warmup: 2 s
time: 5 s
memory time: 0 ns
reduction time: 0 ns
parallel: 1
inputs: none specified
Estimated total run time: 21 s

Benchmarking menet kozben, Enum.take ...
Benchmarking menet kozben, sajat fv ...
Benchmarking utolag keres ...
Calculating statistics...
Formatting results...

Name                        ips        average  deviation         median         99th %
menet kozben, sajat fv      0.97        1.03 s    ±11.20%        0.98 s        1.24 s
menet kozben, Enum.take     0.42        2.37 s     ±5.62%        2.30 s        2.53 s
utolag keres              0.0828       12.07 s     ±0.00%       12.07 s       12.07 s

Comparison:
menet kozben, sajat fv      0.97
menet kozben, Enum.take     0.42 - 2.30x slower +1.34 s
utolag keres              0.0828 - 11.69x slower +11.04 s

:ok
```

Kihagy-bevesz (include-exclude) rekurzió

Kombinációk

```
defmodule Kombinaciok do
  def komb(ns), do: komb(ns, [])

  defp komb([n | ns], acc) do
    komb(ns, acc) ++ komb(ns, [n | acc])
  end
  defp komb([], acc), do: [acc]
end
```

```
{:module, Kombinaciok, <<70, 79, 82, 49, 0, 0, 7, ...>>, {:komb, 2}}
```

```
Kombinaciok.komb([1,2,3]) |> Enum.sort
```

```
[[], [1], [2], [2, 1], [3], [3, 1], [3, 2], [3, 2, 1]]
```

Összeg testvéries elosztása

Adott pénzérméket úgy kell elosztani két ember között, hogy a két összeg különbségének abszolút értéke a lehető legkisebb legyen (CEOI'1995 versenyfeladat).

Legyen ez az érmék értékét tartalmazó lista: [28, 7, 11, 8, 9, 7, 27]. Ekkor egyikük a [9, 11, 28] érméket kapja, melyek összege 48, másikuk a többit, melyek összege 49.

```
defmodule ElosztS do

  def sort(ls), do: Enum.sort(ls, fn (a,b) -> b < a end)

  @type p_int() :: integer()
  @type my_map() :: %{[p_int()] => p_int()}
  @spec max_osszegek(map :: my_map()) :: resmap :: my_map()
  def max_osszegek(map) do
    maxval = Enum.max(Map.values(map))
    for {k,v} <- map, v == maxval, into: %{}, do: {k, v}
  end
end
```

```
{:module, ElosztS, <<70, 79, 82, 49, 0, 0, 10, ...>>, {:max_osszegek, 1}}
```

```
defmodule Eloszt do

  @spec eloszt(vals :: [EnumS.p_int()]) :: map :: EnumS.my_map
    # Legyen halfsum a vals pozitív egészlista összegének a fele (egész osztással).
    # A map kulcs-érték párjaiban a kulcsok vals olyan max. összegű részlistái,
    # melyek összege nem nagyobb halfsum-nál, az értékek pedig e részlisták összege.

  def eloszt([_,_|_] = vals) do
    tot = Enum.sum(vals)
    |> IO.inspect(label: "Listaösszeg")
    tgt = div(tot, 2)
    |> IO.inspect(label: "Célérték")

    # vals = vals #Enum.sort(vals) #my_sort(vals) # rendezzük az értéklistát csökkenő sorrendben
    # |> IO.inspect()
    eloszt(Map.new(), vals, tgt, [], 0)
    # |> IO.inspect()
    |> ElosztS.max_osszegek()
  end
  def eloszt(_), do: "A listának legalább kételeműnek kell lennie."

  @spec eloszt(map  :: EnumS.my_map(),  # map-be gyűjtjük a részlistákat és összegüket
               vals :: [EnumS.p_int()], # a még feldolgozandó érmelista
               tgt  :: EnumS.p_int(),   # a célösszeg (nem változik)
               curr :: [EnumS.p_int()], # a már összegyűjtött részlista
               sum  :: EnumS.p_int())   # a már összegyűjtött részlista összege
             :: resmap :: EnumS.my_map()   # a bővített map, a fv. eredménye

  defp eloszt(map, [val|vals], tgt, curr, sum) do
    curr_new = [val | curr] # az aktuáliss érmével bővített részlista
    sum_new = sum + val     # az aktuáliss érmével megnövelt összeg

    # ha az új összeg nem nagyobb a célösszegnél, berakjuk a map-be, ha kisebb, nem
    # figyeljük meg, hogyan használjuk a pipe-ot az esetleg bővített map továbbadására
    (if sum_new <= tgt, do: Map.put(map, curr_new, sum_new), else: map)
    |> eloszt(vals, tgt, curr_new, sum_new) # 1. ág: val-t bevesszük
    |> eloszt(vals, tgt, curr, sum) # 2. ág: val-t kihagyjuk
  end
  defp eloszt(map, [], _tgt, _curr, _sum), do: map
end
```

```
{:module, Eloszt, <<70, 79, 82, 49, 0, 0, 12, ...>>, {:eloszt, 5}}
```

```
Eloszt.eloszt([28, 7, 11, 8, 9, 7, 27])
```

```
Listaösszeg: 97
Célérték: 48
```

```
%{[9, 11, 28] => 48}
```

```
Eloszt.eloszt([1,2,3,4,5]) |> IO.inspect()
Eloszt.eloszt([4,1,2,5,3]) |> IO.inspect()
Eloszt.eloszt([4,1,2,5,6,3,7]) |> IO.inspect()
```

```
Listaösszeg: 15
Célérték: 7
%{[4, 2, 1] => 7, [4, 3] => 7, [5, 2] => 7}
Listaösszeg: 15
Célérték: 7
%{[2, 1, 4] => 7, [3, 4] => 7, [5, 2] => 7}
Listaösszeg: 28
Célérték: 14
%{
  [3, 5, 2, 4] => 14,
```

```
  [3, 6, 1, 4] => 14,
  [3, 6, 5] => 14,
  [6, 5, 2, 1] => 14,
  [7, 2, 1, 4] => 14,
  [7, 3, 4] => 14,
  [7, 5, 2] => 14,
  [7, 6, 1] => 14
}
%{
  [3, 5, 2, 4] => 14,
  [3, 6, 1, 4] => 14,
  [3, 6, 5] => 14,
  [6, 5, 2, 1] => 14,
  [7, 2, 1, 4] => 14,
  [7, 3, 4] => 14,
  [7, 5, 2] => 14,
  [7, 6, 1] => 14
}
```

Gyakorló feladat: ne tartsuk meg az összes részlistát, csak a korábbiaknál nagyobb összegűeket.