Tartalom

Fibonacci

```
Mix.install([
{:benchee, "~> 1.3"},
# {:benchee_html, "~> 1.0"},
# {:benchee_json, "~> 1.0"},
# {:kino_vega_lite, "~> 0.1.13"},
{:kino_explorer, "~> 0.1.25"},
{:arrays, "~> 2.1"}
])
```

Fibonacci hét változatban

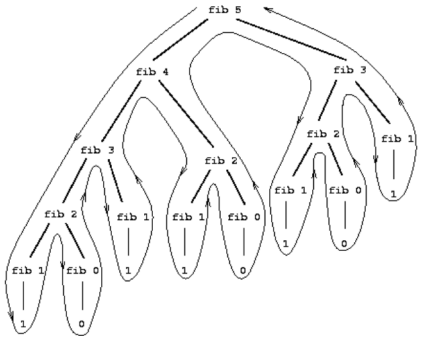
A Fibonacci-számok jól ismert matematikai definíciója:

$F_0 = 0$ $F_1 = 1$ $F_i = F_{i-2} + F_{i-1}$, ha $i > 1$

Naív rekurzív megoldásunk ezt követi.

Az i-edik Fibonacci-szám meghatározása elágazó rekurzióval nagyon rossz hatékonyságú, mert a két elágazó ágat minden egyes rekurzív lépésben újra meg újra teljesen be kell járni, azaz az i-ediknél kisebb Fibonacci-számokat újra és újra ki kell számolni, ráadásul a részeredményeket az egyre mélyülő veremben kell tárolni.

Fib.fib(5) kiszámításának folyamata:



`fib 5`-öt `fib 4` és `fib 3`, `fib 4`-et `fib 3` és `fib 2` kiszámításával stb. kapjuk.

```
defmodule Fib do

  # Tree recursion
  # O(2^n) futási idő, O(2^n) tárhely
  @spec fib(i :: integer()) :: n :: integer()
  # n az i-edik Fibonacci-szám
  def fib(0), do: 0
  def fib(1), do: 1
  def fib(i), do: fib(i-1) + fib(i-2)

end
Fib.fib(23) #|> IO.inspect()
```

28657

```
defmodule FibM do

  # Memoization (top down) – dinamikus programozás
  # O(n) futási idő, O(n) tárhely
  @spec fib_mem(i :: integer()) :: n :: integer()
  # n az i-edik Fibonacci-szám
  def fib_mem(i), do: fib_m(i, %{0 => 0, 1 => 1}) |> elem(0)

  @type mem() :: %{index :: integer() => value :: integer()}
```

```
  @spec fib_m(i :: integer(), mem :: mem()) :: {n :: integer(), uj_mem :: mem()}
  # n az i-edik Fibonacci-szám

  def fib_m(i, mem) do
    case mem[i] do # case nem váltható ki mintaillesztéssel
      nil ->
        {prev, memp} = fib_m(i-2, mem)
        {curr, memc} = fib_m(i-1, memp)
        val = prev + curr
        {val, Map.put(memc, i, prev+curr)}
      val ->
        {val, mem}
    end
  end

end
FibM.fib_mem(63) #|> IO.inspect()
```

```
6557470319842
```

```
defmodule FibT do

  # Tabulation (bottom-up) – dinamikus programozás
  # O(n) futási idő, O(n) tárhely
  @spec fib_tab(i :: integer()) :: n :: integer()
  # n az i-edik Fibonacci-szám
  def fib_tab(i), do: fib_t(i, 2, %{0 => 0, 1 => 1})

  @type tab() :: %{index :: integer() => value :: integer()}
  @spec fib_t(i :: integer(), j :: integer(), tab :: tab()) :: n :: integer()
  # n az i-edik Fibonacci-szám
  def fib_t(i, j, tab) when i < j, do: tab[i]
  def fib_t(i, j, tab) do
    tab0 = Map.put(tab, j, tab[j-2] + tab[j-1])
    fib_t(i, j+1, tab0)
  end
end
FibT.fib_tab(63) #|> IO.inspect()
```

```
6557470319842
```

```
defmodule FibAerl do

  # Tabulation (bottom-up) – dinamikus programozás
  # O(n) futási idő, O(n) tárhely
  # Erlang :array
  @spec fib_tab(i :: integer()) :: n :: integer()
  # n az i-edik Fibonacci-szám
  def fib_tab(i), do: fib_t(i, 2, :array.set(1,1,(:array.set(0,0,:array.new()))))

  @type tab(integer) :: :array.array(integer, integer)
  @spec fib_t(i :: integer(), j :: integer(), tab :: tab(integer())) :: n :: integer()
  # n az i-edik Fibonacci-szám
  def fib_t(i, j, tab) when i < j, do: :array.get(i, tab)
  def fib_t(i, j, tab) do
    prev = :array.get(j-2, tab)
    curr = :array.get(j-1, tab)
    tab0 = :array.set(j, prev+curr, tab)
    fib_t(i, j+1, tab0)
  end
end
FibAerl.fib_tab(1023) #|> IO.inspect()
```

```
278529355069959292393881241266809350935330735212370380691318266898736950320346518362561675961332445274995854966996688219
```

```
defmodule X do
```

```
  @type array(int) :: :array.array(int)

end
```

```
{:module, X, <<70, 79, 82, 49, 0, 0, 5, ...>>, :ok}
```

```
defmodule FibAex do

  # Tabulation (bottom-up) – dinamikus programozás
  # O(n) futási idő, O(n) tárhely
  # Elixir Array
  @spec fib_tab(i :: integer()) :: n :: integer()
  # n az i-edik Fibonacci-szám
  def fib_tab(i), do: fib_t(i, 2, Arrays.new([0,1]))

  @type tab(integer) :: :array.array(integer)
  @spec fib_t(i :: integer(), j :: integer(), tab :: tab(integer)) :: n :: integer()
  # n az i-edik Fibonacci-szám
  def fib_t(i, j, tab) when i < j, do: Arrays.get(tab, i)
  def fib_t(i, j, tab) do
    prev = Arrays.get(tab, j-2)
    curr = Arrays.get(tab, j-1)
    tab0 = Arrays.append(tab, prev+curr)
    fib_t(i, j+1, tab0)
  end
end
FibAex.fib_tab(1023) #|> IO.inspect()
```

```
278529355069959292393881241266809350935330735212370380691318266898736950320346518362561675961332445274995854966996688219
```

```
defmodule FibLtab do

  # Tabulation (bottom-up) – dinamikus programozás
  # O(n) futási idő, O(n) tárhely
  # Elixir List
  @spec fib_tab(i :: integer()) :: n :: integer()
  # n az i-edik Fibonacci-szám
  def fib_tab(i), do: fib_t(i, 2, [1,0])

  @type tab(integer) :: :array.array(integer)
  @spec fib_t(i :: integer(), j :: integer(), tab :: tab(integer)) :: n :: integer()
  # n az i-edik Fibonacci-szám
  def fib_t(i, j, tab) when i < j, do: hd(tab)
  def fib_t(i, j, tab) do
    prev = hd(tl(tab))
    curr = hd(tab)
    tab0 = [prev+curr | tab]
    fib_t(i, j+1, tab0)
  end
end
FibLtab.fib_tab(63) #|> IO.inspect()
```

```
6557470319842
```

```
defmodule FibI do
  # Space optimized (bottom up)
  # O(n) futási idő, O(1) tárhely
  @spec fib_iter(i :: integer()) :: n :: integer()
  # n az i-edik Fibonacci-szám
  def fib_iter(i), do: fib_i(i, 1, 0)

  @spec fib_i(i :: integer(), curr :: integer(), prev :: integer())
    :: n :: integer()
  # n az i-edik Fibonacci-szám
  defp fib_i(0, _curr, prev), do: prev
```

```
  defp fib_i(1, curr, _prev), do: curr
  defp fib_i(i, curr, prev), do: fib_i(i-1, prev+curr, curr)
end
FibI.fib_iter(2203) #|> IO.inspect()
```

```
1122758802217805139807062374577053774698103216103328357864188914950437190254759573354894973127917403652055351021118529152
```

```
Benchee.run(
  %{
    "fib tree recursive" => fn -> Fib.fib(33) end,
    "fib memoization" => fn -> FibM.fib_mem(33) end,
    "fib tabulation" => fn -> FibT.fib_tab(33) end,
    "fib tabula_array_erl" => fn -> FibAerl.fib_tab(33) end,
    "fib tabula_array_ex" => fn -> FibAex.fib_tab(33) end,
    "fib tabula_list" => fn -> FibLtab.fib_tab(33) end,
    "fib iterative" => fn -> FibI.fib_iter(33) end
  },
  # formatters: [
  #  Benchee.Formatters.Console,
  #  {Benchee.Formatters.HTML, file: "fibonacci_bench.html"}    # grafikonok
  #  {Benchee.Formatters.JSON, file: "fibonacci_bench.json"}  # opcionális
  # ],
  profile_after: false
  )
:ok
```

```
Warning: the benchmark fib iterative 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 fib memoization 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 fib tabula_array_erl 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 fib tabula_array_ex 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 fib tabula_list 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 fib tabulation 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 fib tree recursive 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: 49 s

Benchmarking fib iterative ...
Benchmarking fib memoization ...
Benchmarking fib tabula_array_erl ...
Benchmarking fib tabula_array_ex ...
Benchmarking fib tabula_list ...
Benchmarking fib tabulation ...
Benchmarking fib tree recursive ...
Calculating statistics...
Formatting results...

Name                      ips        average  deviation         median         99th %
fib iterative          1677.45 K        0.60 µs  ±7934.35%        0.44 µs        0.89 µs
fib tabula_list        1143.84 K        0.87 µs  ±4153.56%        0.69 µs        1.51 µs
fib tabula_array_erl    180.83 K        5.53 µs   ±264.61%        4.97 µs       10.26 µs
fib tabulation          157.79 K        6.34 µs   ±215.84%        5.66 µs       12.96 µs
fib memoization         147.87 K        6.76 µs   ±215.94%        6.00 µs       13.66 µs
fib tabula_array_ex      79.00 K       12.66 µs    ±96.51%       11.56 µs       23.89 µs
fib tree recursive       0.0185 K    53956.38 µs     ±2.59%    53938.80 µs    58288.08 µs

Comparison:
fib iterative          1677.45 K
fib tabula_list        1143.84 K - 1.47x slower +0.28 µs
fib tabula_array_erl    180.83 K - 9.28x slower +4.93 µs
fib tabulation          157.79 K - 10.63x slower +5.74 µs
fib memoization         147.87 K - 11.34x slower +6.17 µs
fib tabula_array_ex      79.00 K - 21.23x slower +12.06 µs
fib tree recursive       0.0185 K - 90509.23x slower +53955.78 µs
```

```
:ok
```

```
# Módosított változat a memoizálási lépések követésére
defmodule FibMm do
  @spec fib_mem(i :: integer()) :: mem :: %{integer() => integer()}
  def fib_mem(i), do: FibM.fib_m(i, %{0 => 0, 1 => 1}) |> elem(1)
  end
FibMm.fib_mem(5)
```

```
%{0 => 0, 1 => 1, 2 => 1, 3 => 2, 4 => 3, 5 => 5}
```

```
# Interaktív bemenet (n slider); önálló cellába kell rakni
cell = Kino.Input.number("Fibonacci index", default: 10, min: 0, max: 35)
```

```
# Bemenet beolvasása
index =
  cell
  |> IO.inspect(label: "Kino input cell")
  |> Kino.Input.read()
  |> IO.inspect(label: "Kino input read")
```

```
Kino input cell: %Kino.Input{
  ref: "p476tnlhyud6wikh4h7dsiw4yzqqrhot",
```

```
  id: "131674189",
  destination: {Kino.SubscriptionManager,
   :"livebook_x5hje53k--xgga5nyt@127.0.0.1"},
  attrs: %{
    default: 10,
    label: "Fibonacci index",
    type: :number,
    debounce: :blur
  }
}
Kino input read: 10
```

```
10
```

```
# Táblázat
mem = FibMm.fib_mem(index)
Explorer.DataFrame.new(Enum.map(mem, fn {k, v} -> %{index: k, value: inspect(%{k => v})} end))
```

```
#Explorer.DataFrame<
  Polars[11 x 2]
  index s64 [0, 1, 2, 3, 4, ...]
  value string ["%{0 => 0}", "%{1 => 1}", "%{2 => 1}", "%{3 => 2}", "%{4 => 3}", ...]
>
```

```
defmodule FibTdbg do

  # Tabulation (bottom-up) – dinamikus programozás
  # O(n) futási idő, O(n) tárhely
  @spec fib_tab(i :: integer()) :: n :: integer()
  # n az i-edik Fibonacci-szám
  def fib_tab(i), do: fib_t(%{0 => 0, 1 => 1}, 2, i)

  @type fib() :: %{index :: integer() => value :: integer()}
  @spec fib_t(mem :: fib(), j :: integer(), i :: integer()) :: n :: integer()
  # n az i-edik Fibonacci-szám
  def fib_t(tab, j, i) when j > i, do: tab[i]
  def fib_t(tab, j, i) do
    tab
    |> Map.put(j, tab[j-1] + tab[j-2])
    |> fib_t(j+1, i)
    |> dbg()
  end
end
FibTdbg.fib_tab(8) #|> IO.inspect()
```

```
21
```