

Actividad Integradora 5.3 Resaltador de sintaxis paralelo (evidencia de competencia)

Jonathan Josafat Vázquez Suárez - A01734225

La complejidad del algoritmo tomado desde secuencial es considerado lineal, ya que a pesar de hacer una gran cantidad de ejecuciones los ciclos no se anidan ni existen condiciones complejas, al traspasar esto a paralelo tenemos la misma cantidad de ejecuciones pero ejecutada en diferentes núcleos, lo que reduce el tiempo de espera para realizar el próximo trabajo pero también requiere tener un buen control de esa memoria compartida para evitar problemas de modificaciones en archivos paralelos.

Los tiempos obtenidos durante la ejecución del programa en modo paralelo y secuencial es muy diferente, es razonable debido a que el trabajo en paralelo es repartido entre los diferentes núcleos y permite acelerar los tiempos de ejecución, teniendo tiempos de 1.73s vs 6.30s. A pesar de generar una mayor carga de trabajo en paralelo esta diferencia en tiempos es considerablemente beneficiosa.

La programación en elixir requiere de una lógica muy diferente a la aplicada en otros lenguajes, la manera de trabajo al ser un lenguaje de menor nivel complica utilizar funciones predefinidas por el sistema para el manejo de estructuras, pero el pattern matching es realmente una gran utilidad durante la codificación. la utilización de dependencias que permitan el trabajo paralelo como Task, GenServer y algunas otras son una excelente manera de adentrarse en la verdadera profundidad y belleza del lenguaje. A pesar de las funcionalidades únicas de elixir, este lenguaje igualmente permite aplicar técnicas de codificación en otros lenguajes de mayor nivel, acelerando estos como ejemplo al utilizar funciones por pattern matching en lugar de una serie de ifs anidados.

Velocidades de ejecución:

```
Benchmarking Paralelo...
Benchmarking Secuencial...
```

Name	ips	average	deviation	median	99th %
Paralelo	0.58	1.73 s	±3.36%	1.70 s	1.80 s
Secuencial	0.159	6.30 s	±0.00%	6.30 s	6.30 s

Ejecución benchmarking:

```
iex(2)> Concurrency.init
Operating System: Linux
CPU Information: AMD Ryzen 7 4800H with Radeon Graphics
Number of Available Cores: 16
Available memory: 15.36 GB
Elixir 1.11.4
Erlang 22.2.7

Benchmark suite executing with the following configuration:
```

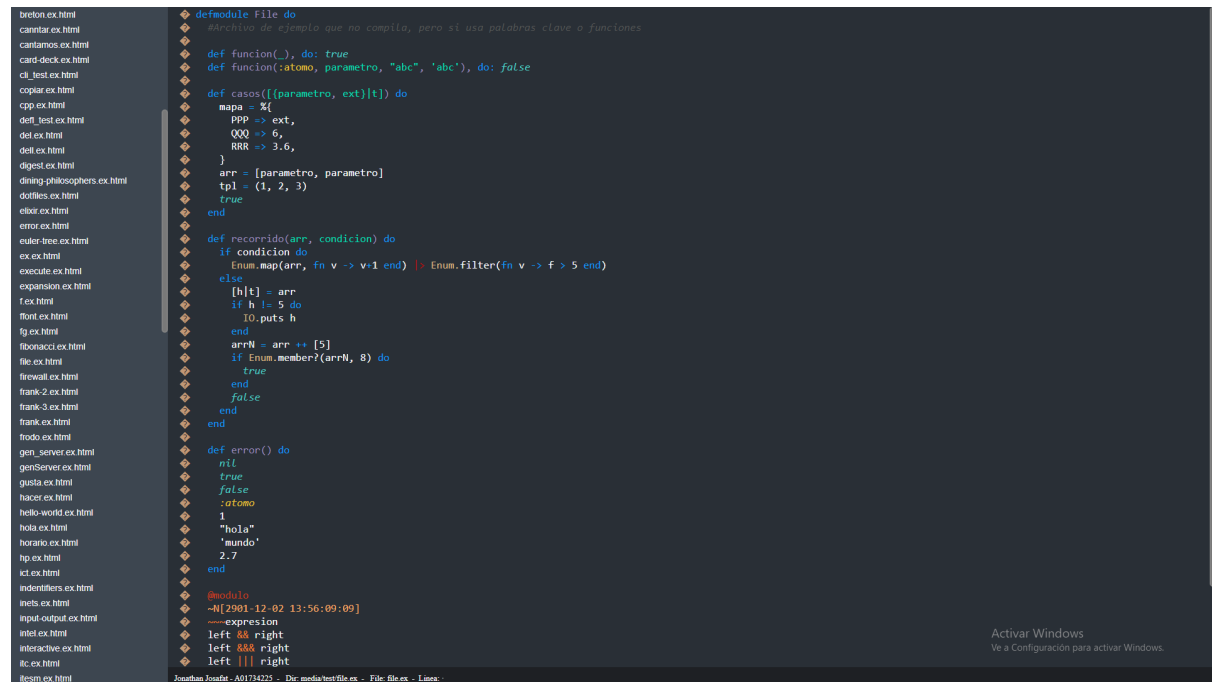
Ejecución paralela/secuencial:

[illegible]

Ejecución del front:

```
password@LAPTOP-9PVBIEED:/mnt/c/cod/elixir/concurrencia/web$ http-server
Starting up http-server, serving ./
Available on:
  http://127.0.0.1:8080
  http://192.168.1.70:8080
```

Visualización de resultados:



The screenshot shows a code editor with a file explorer on the left and Elixir code in the main area. The file explorer lists various .ex.html files. The code in the main area is an Elixir module named 'File.do' with several functions and a test case. The code is as follows:

```
defmodule File do
  # Archivo de ejemplo que no compila, pero si usa palabras clave o funciones

  def funcion(_, do: true)
  def funcion(:atomo, parametro, "abc", 'abc'), do: false

  def casos([([parametro, ext])|t]) do
    mapa = %{
      PPP -> ext,
      QQQ -> 6,
      RRR -> 3.6,
    }

    arr = [parametro, parametro]
    tpl = (1, 2, 3)
    true
  end

  def recorrido(arr, condicion) do
    if condicion do
      Enum.map(arr, fn v -> v+1 end) |> Enum.filter(fn v -> v > 5 end)
    else
      [h|t] = arr
      if h != 5 do
        IO.puts h
      end
      arrN = arr ++ [5]
      if Enum.member?(arrN, 8) do
        true
      else
        false
      end
    end
  end

  def error() do
    nil
    true
    false
    :atomo
    1
    "hola"
    "aundo"
    2.7
  end

  @moduledoc ~N[2981-12-02 13:56:09:09]
  ~>expresion
  left && right
  left &&& right
  left ||| right
end
```

Directorio archivos resultados:

