**Actividad Integradora 5.3: Resaltador de sintaxis paralelo**

A través del algoritmo de distribución y ejecución de procesos en paralelo implementado para la solución de esta actividad se pretende mejorar el tiempo de ejecución del código cuando tiene como entrada muchos archivos. Así mismo, se busca comparar el desempeño de dicho algoritmo en paralelo contra su alternativa común, la ejecución iterativa.

Primeramente, analizando la complejidad de la solución planteada, observamos que la transición y lectura del archivo de Leex de cada carácter del código en Javascript que recibe como entrada es de orden O(n), debido a que todas estas expresiones regulares que realizan esta tarea son un autómata finito determinista. Esto nos genera un conjunto m de tokens, que es menor en cantidad en la mayoría de los casos que los n caracteres de entrada.

Luego, nuestra función en el programa principal de Elixir que convierte los tokens a HTML, al tener que pasar por cada token y convertirlo a código web, tendría una complejidad lineal de O(m). Como hicimos 2 pasos de orden lineal en ese proceso, podríamos decir que obtenemos una complejidad de O(2n). Sin embargo, como 2n sigue siendo una constante, se puede representar como una solo n. Con esto, el algoritmo en general tendría una complejidad lineal de O(n).

Ahora bien, nuestra función que permite dividir la tarea de convertir los archivos de entrada en procesos que harían diferentes cores de la PC, idealmente tendría una complejidad de O(n/c), dónde c sería una constante que represente el número de procesadores que se emplearán. Sin embargo, al ser dicha división otra constante, podemos decir que la complejidad se mantendría lineal, es decir, O(n).

Después de implementar la solución a esta actividad y de hacer las pruebas midiendo los tiempos de varias ejecuciones, se pudo observar como claramente el algoritmo en paralelo es más eficiente y veloz que el iterativo, por la correcta distribución de tareas que se realizó. A continuación se muestra un ejemplo de ejecución del benchmark:

```
Benchmarking iterative...
Benchmarking parallel...

Name                ips        average  deviation         median         99th %
parallel           4.46      224.17 ms     ±4.99%      219.00 ms      266.00 ms
iterative          2.26      442.67 ms     ±5.52%      437.00 ms      516.00 ms

Comparison:
parallel           4.46
iterative          2.26 - 1.97x slower +218.49 ms
```

Si el tiempo de ejecución hubiera sido mayor en el algoritmo paralelo, esto sería debido a que la entrada del programa no es lo suficientemente pesada para que valga la pena hacerlo en paralelo. Además, el emplear un procedimiento en paralelo también involucra la creación de los procesos con su información correspondiente, tiene que registrarlos en su tabla, activar el proceso y estarlo monitoreando, lo que en inglés se le conoce como house keeping.

En cuanto a las implicaciones éticas de este algoritmo, considero que ahora resulta mucho más eficiente y conveniente de usar con la implementación de la distribución de tareas en los distintos procesadores de la PC, ya que así se ejecutará con mayor velocidad el algoritmo cuando tenga entradas de muchos archivos. En el caso de que un algoritmo de este estilo se implementara en alguna empresa, esto permitirá que no haya pérdidas económicas fuertes que impacten negativamente a las personas involucradas ya que se ejecutaría eficientemente realizando los procesos requeridos por la empresa en tiempo y forma.

Como conclusión, me gustaría destacar lo interesante que me pareció este tema de la distribución de procesos. Antes de aprenderlo en esta unidad de formación, no tenía idea de que este tipo de procedimientos se podían realizar con software, por lo que me resultó muy interesante y útil, por todas las ventajas que se mencionaron previamente.